NAVAL POSTGRADUATE SCHOOL
Monterey, California

DISSERTATION

SOLUTION OF LARGE-SCALE ALLOCATION PROBLEMS WITH PARTIALLY OBSERVABLE OUTCOMES

by

Kirk A. Yost

September 1998

Dissertation Supervisor: Alan R. Washburn

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 1998	**3. REPORT TYPE AND DATES COVERED** Doctoral Dissertation

4. TITLE AND SUBTITLE **Solution of Large-Scale Allocation Problems with Partially Observable Outcomes**	**5. FUNDING NUMBERS**
6. AUTHOR(S) Yost, Kirk A.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

We develop methods for optimally solving problems that require allocating scarce resources among activities that either gather information on a set of objects or take actions to change their status. Also, the information we gather on the outcomes of the actions we take may be erroneous. The latter situation is called *partial observability*, and methodology available prior to this dissertation is combinatorially intractable for problems with more than one object. We use two previously-uncombined methods — linear programming (LP) and partially observable Markov decision processes (POMDPs) — to construct a decomposition procedure to solve the resulting large-scale allocation problem with partially observable outcomes. We show theoretically that this procedure is both optimal and finite; in addition, we develop improvements to the procedure that reduce runtimes on test problems by 95%. We demonstrate the procedure on a small targeting problem with a known analytical solution, as well as a large-scale military example concerned with allocating aircraft sorties, weapons, and bomb-damage assessment sensors to targets. Finally, we develop analytical bounds on the expected objective function values of a related allocation problem with more stringent resource constraints, and present a simulation-based approach to estimate the distributions of the outcomes for that model.

14. SUBJECT TERMS POMDP, MDP, Linear Programming, USAF, BDA, sensor modeling	**15. NUMBER OF PAGES** 190
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

SOLUTION OF LARGE-SCALE ALLOCATION PROBLEMS WITH PARTIALLY OBSERVABLE OUTCOMES

Kirk A. Yost
Lieutenant Colonel, United States Air Force
B.S., United States Air Force Academy, 1980
M.S., Rensselaer Polytechnic Institute, 1986

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN OPERATIONS RESEARCH
from the

NAVAL POSTGRADUATE SCHOOL
September 1998

Author: [illegible]
Kirk A. Yost

Approved by: [illegible]
Alan R. Washburn
Professor of Operations Research
Dissertation Supervisor

[illegible]
Gerald G. Brown
Professor of Operations Reseach

[illegible]
Robert F. Dell
Associate Professor of Operations Reseach

Guillermo Owen
Guillermo Owen
Professor of Mathematics

[illegible]
Craig W. Rasmussen
Associate Professor of Mathematics

Approved by: Richard E. Rosenthal
Richard E. Rosenthal, Chair, Department of Operations Research

Approved by: Maurice D. Weir
Maurice D. Weir, Associate Provost for Instruction

ABSTRACT

We develop methods for optimally solving problems that require allocating scarce resources among activities that either gather information on a set of objects or take actions to change their status. Also, the information we gather on the outcomes of the actions we take may be erroneous. The latter situation is called *partial observability*, and methodology available prior to this dissertation is combinatorially intractable for problems with more than one object. We use two previously-uncombined methods — linear programming (LP) and partially observable Markov decision processes (POMDPs) — to construct a decomposition procedure to solve the resulting large-scale allocation problem with partially observable outcomes. We show theoretically that this procedure is both optimal and finite; in addition, we develop improvements to the procedure that reduce runtimes on test problems by 95%. We demonstrate the procedure on a small targeting problem with a known analytical solution, as well as a large-scale military example concerned with allocating aircraft sorties, weapons, and bomb-damage assessment sensors to targets. Finally, we develop analytical bounds on the expected objective function values of a related allocation problem with more stringent resource constraints, and present a simulation-based approach to estimate the distributions of the outcomes for that model.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

A particularly vexing problem facing today's military is that of bomb-damage assessment (BDA). Certain targets, such as buried command bunkers and industrial complexes, are very difficult to assess after an attack. The introduction of long-range weapons has further complicated the BDA problem, as we frequently must assess the state of a target from hundreds of miles away. As was documented during the Gulf War, such assessments are often erroneous.

The demands of a large-scale campaign, coupled with potentially erroneous assessments, make the problem of allocating strikes to targets even more difficult. Each target may have a different probability of surviving depending on the attacks we have previously allocated to it, and we have constraints on the availability of attack resources and information resources (e.g., aircraft sorties and weapons, and satellite imagery). A second strike or assessment on a target costs additional resources, and is wasteful if the first strike succeeded.

To date, the US Air Force has relied on linear programming (LP) to determine the appropriate mix of weapons to procure. The LPs in use, however, assume perfect knowledge of the outcomes of an attack, or else schedule attacks independently of any information gathered by BDA sensors. These assumptions make it impossible to use the existing models to analyze the larger system problem: that of determining the best mix of attack and BDA assets. The two types of assets are not independent, but it has not been clear how model the interactions of these assets in a unified optimization.

The question of optimizing the investment in transformation (attack) and information (BDA) assets is of crucial importance to the US Department of Defense. Most advanced weapons and sensor systems are enormously expensive, and they are closely related. We need information to allocate weapons effectively, but we cannot attack a target with information alone.

Consequently, the accuracy and timeliness of the information is as important as the effectiveness of the weapon, and while the literature contains many models which model the latter in tremendous detail, there are almost no methodologies to measure the effects of the former. This problem motivates this dissertation.

The condition of imperfect information observation is known in the Operations Research literature as *partial observability*, and is often modeled using an extension of the theory of Markov Decision Processes (MDPs). The partially observable Markov Decision Process (POMDP), however, can only be solved for control of a single object and requires that we put explicit marginal prices on resources. The optimal solution to a POMDP is a "policy," that is, a rule for taking actions depending on the state of the object.

A policy is essentially the optimal decision tree for a particular set of rewards and resource costs, and we can compute the expected consumption of resources and the expected total payoff for any given policy. If the number of policies were small, then we could use them in an LP directly and solve the problem. Unfortunately, the number of policies even for a single object over a short time horizon is combinatorially explosive. For example, the number of possible attack and BDA policies for a single target with 5 available weapon types, 2 sensor types, 2 states (target live or dead), and 5 decision periods is 4.47×10^{16}. As a result, we have the following situation: we cannot use LP, because there are too many possible policies. We also cannot use the POMDP, because it requires prices on resources and cannot handle resource constraints.

Nevertheless, we can combine the two methods to solve the large-scale allocation problem with partially observable outcomes. The LP, such as those long used by the US Air Force, implicitly computes prices on resources via marginal costs. Therefore, the POMDP can use these costs to determine an optimal "policy" for allocating resources to a target based on the probability it is currently dead, its value in the campaign, and the implicit resource costs. The resulting *dynamic column generation algorithm* is the key idea in this dissertation; the LP allocates policies to targets, while the POMDP subproblems determine policies that potentially improve the solution. Figure 1 below shows the algorithm:

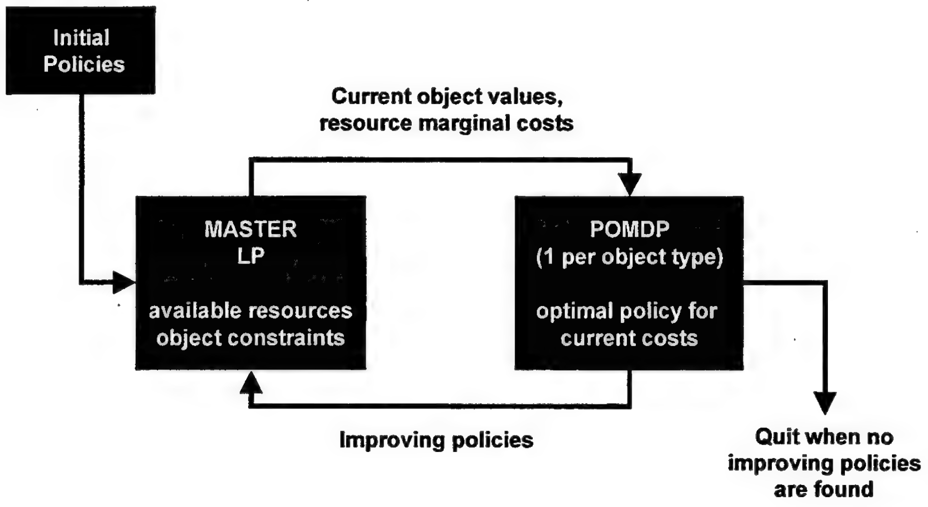
Initial Policies
Current object values, resource marginal costs
MASTER LP
available resources object constraints
POMDP (1 per object type)
optimal policy for current costs
Improving policies
Quit when no improving policies are found

Figure 1: Basic Decomposition Algorithm. The master LP determines marginal resource costs for the current set of policies, and passes them to the POMDPs. The POMDPs use those costs to determine improving policies. The algorithm ends when no improving policies can be generated.

We show that this algorithm is theoretically convergent and finite, and we also develop computational improvements that reduce the algorithm's runtime by 95% on representative test problems. In the case of the large-scale campaign problem above, we solve an example with 9 attack aircraft types, 65 weapon types, 6,313 total targets, and a 9-period time horizon on a typical personal computer in less than 2 minutes.

The algorithm above requires that we constrain resource consumptions in expectation only, so it is possible that the solutions recommended by the optimization may use more resources than are available in certain instances. When such violations are allowed, we refer to the allocation problem as "soft".

Nevertheless, the methodology we develop here also can be used to solve a "rigid" allocation problem, where resource constraints must hold under all circumstances. We prove that solving the soft version of a rigid problem provides an analytical upper bound on the objective function value, and further show how to determine an analytical lower bound on the objective function value. Furthermore, we offer a sequential procedure using the algorithm that successively solves the allocation problem across time, which not only provides better solutions to the rigid problem, but allows estimating the distributions of outcomes via simulation. We test these methods on a targeting problem from the literature that has a known analytical solution, as well as the large-scale campaign example, achieving favorable results in both cases.

Our methodology solves problems that are intractable using techniques previously documented in the optimization and Markov Decision Process literature. By employing two previously-uncombined methods – LP and POMDPs – we can solve both the motivating military problem described above as well as a variety of other applications. In the "information age," it will become increasingly common to encounter situations where we must decide between opportunities for transformation or information with limited resources. We offer methodology that can address a large class of these problems.

ACKNOWLEDGEMENTS

For some reason, it is customary in this section of a dissertation to mention your family last. This makes little sense to me. If the work I present in this document endures, it will be due largely to the foundation provided by my wife Thula and my son Arjuna.

Some years ago, I read an article that defined a dissertation as "a paper written by a professor under difficult circumstances." I defied this claim and wrote this dissertation, but I must admit I was guided with the sort of artistry and subtlety that only comes from four decades of experience. Some day my advisor, Professor Al Washburn, may tell how much of this work was mine and how much was him waiting for me to figure it out; but, it's probably better for me that he remain inscrutable on this subject. I *can* say with certainty that Professor Washburn is both a scholar and educator of the first rank, and I am proud that this document has earned his signature.

I would also like to thank Professors Jerry Brown and Rob Dell, whose pointed lectures on mathematical programming gave me the insights I needed to do this work. I have profited greatly from advice offered by Professor Kevin Wood, who generously offered his time despite the fact that he wasn't on my committee and had no responsibilities for my research. I would also like to acknowledge Dr. Tony Cassandra, whose technical articles on partially observable Markov decision processes offer readability that stands in welcome contrast to rest of the literature in that field. I am not ashamed to admit to the amount of time I spent studying Tony's "POMDPs for Dummies" articles on the Internet.

A host of other professors at NPS made important contributions to my education. In particular, I would like to acknowledge: Guillermo Owen and Craig Rasmussen, the other two members of my committee; Toi Lawphongpanich and Lynn Whitaker, who offered me "out-of-hide" courses on advanced topics; Sam Buttrey, Bob Read, and Kneale Marshall, who worked with me on various subjects when I was getting ready for my comprehensive exams; Dave Morton, who provided detailed and ultimately useful advice on my initial research proposal; Commander Tom Halwachs, USN (Ret), who spends so much time solving other people's hardware and software problems he never gets to concentrate on his own research; Rick Rosenthal, whom I'd worked with prior to becoming a student again and heavily influenced my decision to enter the program at NPS; and Lieutenant Colonel Jeff Appleget (USA) and Lieutenant Colonel Steve Baker (USAF), my predecessors in the Ph.D. program, whose advice on navigating through "the system" was invaluable.

Finally, I would like to thank Mr. Jerry Colyer and Major General Roy Bridges, my bosses at my previous assignment. They not only agreed to release me early for a graduate school assignment but also campaigned vigorously for the various waivers I needed when it was clearly not in their interest to do so. I have been gratified to see that in the last three years they have brought in a great deal of young talent that, in many cases, has done a better job than I would have had I stayed.

I. INTRODUCTION

A. THE MOTIVATING PROBLEM

The "Information Age" that is now so common in the United States probably arrived in military affairs in the 1980s, when we began to acquire in volume conventional weapons whose ranges extended beyond the horizon. A recent textbook notes that

> [Commanders] have demanded (and continue to demand) more data and information, with heavy emphasis on speed and timeliness ... consequently, the critical intelligence analysis and the command and control (C^2) decision and action planning processes have been overwhelmed by information volume... Obviously, a key to this information management problem is the ability to combine or "fuse" data, not only as a volume-reducing strategy, but also as a means to exploit the unique combinations of data that may be available. (White, 1990, pp. xi)

This quote describes much of the current emphasis in the sensor world, where the overriding assumption is that there are externally-generated information requirements, and the main issue is determining the best way to allocate sensors among the requirements and fuse the resulting information.

But this description stops short of the larger problem. Commanders want more information for a reason; they want to use it to make decisions. Since the quality of the decisions is closely related to the quality of the data available, there is now a funding competition between weapons and sensors. The current commander of USSPACECOM, General Howell M. Estes III, commented on this issue:

> Hard choices need to be made between investment in information infrastructure [and] the combat systems themselves. This is an extreme dilemma, because combat systems, without timely, relevant information, are useless. On the other hand, you can't take out an enemy tank with just information. We need to strike a balance between 'shooters' and 'information systems' if we're going to be successful in the future. (Scott 1998)

A particularly vexing information problem facing the military today is that of bomb-damage assessment (BDA). Certain targets, such as the aircraft shelters shown in Figure 1.1, are very difficult to assess correctly. Also, the use of long-range weapons means that some sensor or sensors must perform follow-up assessments to determine if the target is still functioning. Such assessments are subject to error, and BDA proved to be an extraordinarily contentious issue during DESERT STORM (e.g., Lewis 1994). Commanders who assumed that information would automatically be available for attacked targets quickly found that sensor resources were limited, and in some cases required substantial response time.

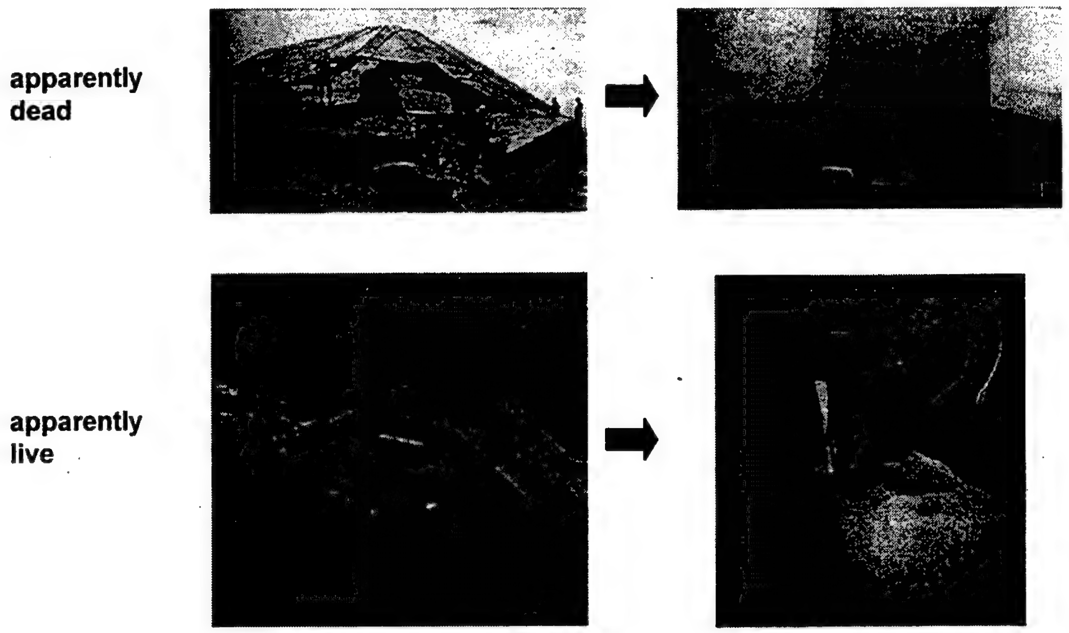

Figure 1.1: Examples of Difficult-to-Assess Targets. These aircraft shelters were bombed during DESERT STORM. The top shelter appears to be heavily damaged, but the interior is pristine. The bottom shelter has an insignificant hole in the roof, but the contents are completely destroyed. (From Cohen 1993, pp. 39-43)

Even when BDA was delivered, the information was often not useful. General Norman Schwartzkopf, the commander of US Central Command, gave an extreme example to the Senate Armed Services Committee in testimony following DESERT STORM. He commented that his J-2 (intelligence) organization rated a 4-span bridge as "50% destroyed" because only two spans were knocked out, despite the fact the bridge was functionally useless (Grundhauser et al. 1993, p. 101).

Recent analyses (e.g., Evans 1996, Aviv and Kress 1997) have shown that the allocation of sensors and the allocation of weapons should not be treated independently. The quality of the allocated BDA directly affects the possibility of an erroneous assessment, which also affects the decision whether to repeat the attack. This "sensor-shooter" allocation is the motivating operational problem for this dissertation, and is formulated as follows:

> **Given a set of targets with a given value structure, a set of weapons with known effectiveness, a set of BDA sensors with known error rates and response times, and a finite time horizon, how do we best allocate these resources to the targets?**

B. GENERAL PROBLEM

There is a general problem as well, and its potential applications extend well beyond the sensor-shooter military problem. This general problem has several important characteristics.

- Discrete Objects, States, and Time: We are trying to control a finite set of objects, each with a known and finite number of states. The time horizon is divided into discrete intervals, with an opportunity to apply some available control to an object in each interval.
- Markovian State Evolution: The states of the objects in the next time interval are a function only of their current state and the current control applied to the objects. In other words, the history of object states and controls applied does not affect the next state.
- Known Additive Rewards: There is a known reward for having each object in each state at the end of the time horizon, and the total reward is the sum of these rewards. While we could address certain nonlinear reward structures, this dissertation considers only linear and additive rewards.
- Fixed Resources: There are fixed amounts of various resources available, and we require that the *expected* resource consumption does not exceed the available resources.
- Discrete Sets of Actions and Stochastic Consumptions: There is a finite set of available actions that can be applied to the objects in any time period. These actions may consume random amounts of the available resources.
- Independent Objects and Independent Random Outcomes: The state of any object is independent of the state of any other object. Also, the outcomes of the actions are random, and are independent among all other actions and object states.
- Partial Observability: The outcomes of the actions are *partially observable*; that is, we cannot observe the state of an object without error. This characteristic (called "noise" in engineering applications) is the distinguishing feature of this problem.

The sensor-shooter problem contains all these characteristics. There are limited numbers of weapons and sensors available, and commanders must allocate them across time. Further, the targets have known values, and their states (say, live or dead) are functions of their current state and the actions taken. Taking an action may consume a random combination of resources; for example, an attacking aircraft may not be able to deliver its full load of bombs, or the aircraft itself may be lost in the attack. The outcomes of the attacks are random as well. Finally, as demonstrated in Figure 1.1, we may be uncertain of the outcomes of the attacks, regardless of the sensor assets used.

As mentioned, this problem extends beyond the military, and applies to any allocation problem where the states of the objects cannot be observed with certainty. In medicine, we must use tests and diagnoses that are subject to error to assess the state of the patient, and the possible treatments also have random effects. In the criminal justice system, we must rely on noisy psychiatric exams and case histories to assess the state of an inmate, and the possible actions (parole, incarceration, various rehabilitation programs) have random effects.

Nevertheless, we must make allocation decisions under these unclear circumstances, and we also must not exceed our available resources. Now, finding a *feasible* solution is not a hard problem; we usually have rules of thumb and heuristics available to determine legitimate allocations. In the sensor-shooter problem, the US Air Force often relies on a "shoot-look-shoot" doctrine, which simply states that we attack, look with a sensor, and attack again if the sensor reports the target is still alive. Restricting the allocations to these types of rules makes it possible to quickly find feasible solutions.

Unfortunately, such rules give us no clear idea of how well we are using our resources. A rule-of-thumb or heuristic solution may be a very good one, but without knowledge of bounds on the rewards, we cannot know if we can improve the allocation of scarce, valuable assets.

Another important issue associated with this problem is that raised by General Estes, which is trading the cost of acquiring information versus the cost of taking actions. Observing an object consumes resources. Sensors cost money, as do visits to a physician, and psychiatric examinations of criminals. These information-gathering activities do not affect the state of the objects, but they have a drastic effect on subsequent actions taken.

C. APPROACHES IN THE LITERATURE

In the operations research literature, there are two paths to search for solutions to the problem stated above. The first path stems from viewing the problem as an allocation of constrained resources. The second path results from viewing the problem as one of finding the best policies, or rules for taking actions based on the time period and the state, for each object. This section does not describe these approaches in detail, as each has a huge literature. The intent is to show that the methodology available on each path can address only part of the problem.

1. Allocation-Based Approaches

Given the additive nature of the problem, it seems reasonable to begin by casting it as a linear program (LP). An LP model is based on three primary axioms (e.g., Dantzig and Thapa 1997, pp. 22-23): proportionality, additivity, and continuity. A fourth assumption originally required by Dantzig, determinism (Dantzig 1963, p. 7), has disappeared in recent years; we speculate this is due to Dantzig's later research interests in stochastic models. Nevertheless, the stochastic nature of the problem does not rule out LP as a solution method.

In our model, let i index the resources in some set I. Assume that the objects can be grouped into a set of J classes, indexed by j, and are indistinguishable within each class. Each object can be controlled by one out of a set of available *policies*. Policies are defined formally in Chapter II, but for now we define a policy as a rule for taking an action based on the history of actions applied to the object, the history of observations, and the current time period. Further, we require that each policy only apply to a single object.

Let S be the set of possible policies, indexed by s. Denote the reward for object j in state $e \in E$ at the end of the time horizon as the vector $r_j = (r_{je})$, and let Y_{sji} be the random amount of resources i consumed by object j under a policy $s \in S$. Let x_{sj} be the number of objects j following policy s, N_j be the number of objects j in each class, and $b_i \geq 0$ be the fixed amount of resources of type $i \in I$ available. Note that the resources may be available across time or only at certain times; the set I includes both cases.

Using these definitions, the resource and allocation constraints are

$$\begin{aligned}
&\sum_{s\in S, j\in J} Y_{sji}\, x_{sj} \leq b_i \quad \forall\, i \in I \\
&\sum_{s\in S} x_{sj} = N_j \quad \forall\, j \in J \qquad (1.1) \\
&x_{sj} \text{ integer} \quad \forall\, s \in S, j \in J.
\end{aligned}$$

These are not legitimate constraints for an LP. The consumption parameters in the resource constraints are random, violating proportionality. Furthermore, the x_{sj}'s are integral, which violates the continuity axiom. Nevertheless, the specifications of the general problem permit the resource constraints to be satisfied on the average, so we can substitute the expected consumptions $E(Y_{sji})$ for the random parameters. In addition, we can redefine x_{sj} to be the *expected* number of applications of policy s to object j. The revised constraints are

$$\begin{aligned}
&\sum_{s\in S, j\in J} E\left(Y_{sji}\right) x_{sj} \leq b_i \quad \forall\, i \in I \\
&\sum_{s\in S} x_{sj} = N_j \quad \forall\, j \in J \qquad (1.2) \\
&x_{sj} \geq 0 \quad \forall\, s \in S, j \in J.
\end{aligned}$$

This constraint set follows proportionality, continuity, and additivity axioms. Now, we must consider the objective function. Under the assumptions of the general problem, the states of the objects we are trying to control are not known with certainty. Therefore, every policy we apply to an object leads to some probability distribution of states at the end of the time horizon. Define $\pi_{sj} = (\pi_{sje})$ as the probability vector for each object j after applying policy s. Then, the expected reward for using policy s on object j is $E(R_{sj}) = r_j \cdot \pi_{sj}$, and we can write the objective function as

$$\max_x \sum_{s \in S, j \in J} E(R_{sj}) x_{sj} . \tag{1.3}$$

This objective function obeys the proportionality and additivity axioms as well, and the use of expected values in the objective is justified by the assumption that the rewards (or utility) are additive over the objects. The combination of (1.2) and (1.3) yields a legitimate LP.

The unspecified part of this LP is the set S of policies. Since these constitute the columns of the LP, we must be concerned with the size of S. Let A be the set of available actions for a particular object. Further, let $O \subseteq A$ be the subset of the possible actions that observe the object and provide information we can use to predict its state; then $A - O$ is the set of actions that may affect the object's state, but do not provide any information.

Using the above definition of a policy, we can write a difference equation to determine the number of possible policies L_t for each object as a function of $|A|$, $|O|$, the number of possible observations $|E|$, and the time period $t = 1, 2, \ldots$. This equation is

$$\begin{aligned} L_1 &= |A| \\ L_t &= |A - O| L_{t-1} + |O| L_{t-1}^{|E|} . \end{aligned} \tag{1.4}$$

The justification for this difference equation is that in the first period, we can choose any of the available actions. After that, however, choosing an action that yields information requires that we specify an action for every possible observation. Since there are L_{t-1} policies available for every observation, we end up with the exponential term in (1.4).

This makes the size of S combinatorially explosive. Table 1.1 shows the number of policies in S for a single-object problem with 5 total actions, 2 observation actions, and 2 states.

Time Period	0	1	2	3	4	5
Possible Policies	1	5	65	8645	1.49E+08	4.47E+16

Table 1.1: Number of Possible Policies by Number of Time Periods for a 4-Action, 2-Observation Problem with 2 Object States. The number of possible policies grows exponentially with time.

The combinatorial nature of the number of policies makes LP untenable except for cases with very short time horizons and small numbers of objects, actions and observations. As an example, consider the sensor-shooter problem. These problems usually contain 9 time periods, have up to 20 different sensors available, and up to 100 different attack options. Directly solving an LP with such an enormous number of columns is not currently possible.

While the number of possible policies is huge, many of them are uneconomical or dominated by other policies. If it were possible to cull the unproductive policies and find only the useful ones, we could reduce the LP to a manageable size and solve the problem. This observation leads to the second path in the operations research literature, the policy-based approaches.

2. Policy-Based Approaches

Since an LP requires the enumeration and solution of a problem with an intractable number of columns, perhaps we should to try to find the best policy or policies for each object.

For a single object, a stochastic decision tree is appealing. A particularly good example of this is given in Marshall and Oliver (1995, pp. 173-179) on the detection of colon cancers. The problem is similar to the general problem in this dissertation; the patient has two states (having colon cancer or not), and he can choose a variety of tests to determine whether he has the disease. The tests may be administered in various sequences, and they have differing costs and error rates. The Marshall and Oliver analysis compares various sequential applications of tests and computes the expected costs of testing regimes versus the expected number of cancers detected.

Unfortunately, stochastic decision trees have difficulties for the general problem. The stochastic tree has the same combinatorially explosive number of nodes as the equation (1.4).

While not impossible, forming the tree and "rolling back" to find the best policy would be computationally challenging.

We can overcome some of the computational difficulties of the stochastic tree by considering stochastic dynamic programming (e.g., Bellman 1957). At every node, the stochastic tree must contain branches for every possible action and every possible random outcome. This, as pointed out above, is untenable for problems we are considering. Stochastic dynamic programming is more efficient, however, because it only requires making a decision for each possible object state in each time period (e.g., Kall and Wallace 1994, p. 129). For objects with a small number of states, this approach seems reasonable.

Furthermore, the Markovian nature of the objects leads to a well-developed body of literature on Markov decision processes (MDPs) (Howard 1960). An MDP is a 4-tuple $\langle E, A, P, R \rangle$, where E is the set of object states, A is the set of available actions, and R is a set of costs and rewards. P defines the transition probabilities of the process; in other words, P describes the probabilities of a state transition given the current state and the action taken.

MDP'*s* are usually solved using stochastic dynamic programming, so it appears we are on the way to a solution for the general problem. We still have to cope with the partial observability issue, however. But, even this has a well-developed body of literature. The basic MDP model was extended by Sondik (1971) to include a noisy observation model. The resulting *partially observable* Markov decision process (POMDP) is a 6-tuple $\langle E, A, P, R, \Theta, B \rangle$. Here Θ is the set of possible observation outcomes, and B is the set of probabilities of a particular observation based on the action taken and the state of the object.

The POMDP model appears to be the appropriate mechanism for sorting through possible policies for each object. POMDPs are also solved using stochastic dynamic programming, and their assumptions match most of the assumptions of the general problem. There is one crucial shortfall: we cannot completely specify R for the POMDP. While we know the terminal rewards for objects in each class, we do not have explicit costs for the resources.

POMDP formulations in the literature (e.g., Monahan 1980, Eagle 1984) assume known marginal costs for the actions, but the general problem does not specify these costs. Instead, we only have constraints on the resources, and we want multiple objects to share common resources. With one important exception that we discuss shortly, POMDP researchers assume known costs and rewards, and only consider control of a single object.

3. Summary of the Available Approaches

Traveling down the two paths, it appears that each approach *almost* solves the problem. If we could a priori generate a manageable number of policies, we could model the general problem as an LP and solve it. Conversely, if we could find a set of costs that would keep expected resource consumption within the constraints, we could formulate and solve a POMDP for each object and follow those policies.

There is one paper in the literature that uses POMDPs to allocate a single common, constrained resource to multiple objects. This work was done by Castenon (1997), and is similar to the sensor-shooter problem. Castenon is concerned with allocating constrained sensor looks to a number of targets over a finite number of time periods. He suggests searching for resource costs that result in policies that, on the average, do not violate the resource constraints, and uses subgradient searches to find these cost parameters. Castenon notes that extending this methodology to a problem with multiple object types and resources – our general problem – is still an open research question.

D. DISSERTATION OVERVIEW

As it turns out, Castenon's work (which we did not discover until our work was substantially complete) is an important stepping-stone to the theory developed in this dissertation. He was the first to recognize that some external search procedure was necessary to find pseudo-costs for the constrained resources. Unfortunately, the methods he suggests are not appropriate for large numbers of shared resources.

Instead, we combine approaches from the two paths – linear programming and partially observable Markov decision processes – into an integrated solution procedure for the general problem. The methodology is simple enough to capture in a single graphic (Figure 1.2).

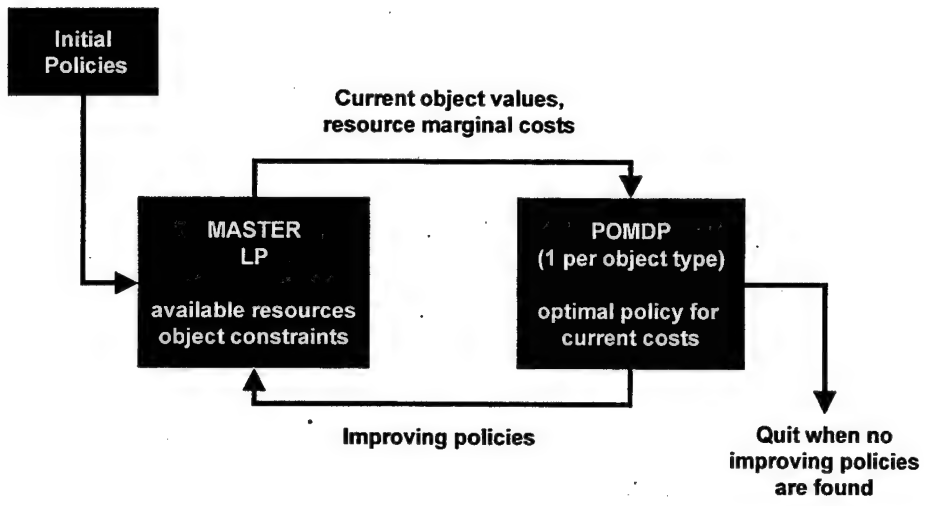
Initial Policies
Current object values, resource marginal costs
MASTER LP
available resources object constraints
POMDP (1 per object type)
optimal policy for current costs
Improving policies
Quit when no improving policies are found

Figure 1.2: Basic Decomposition Algorithm. The master LP determines marginal resource costs for the current set of policies, and passes them to the POMDPs. The POMDPs use those costs to determine improving policies. The algorithm ends when no improving policies can be generated.

The LP cannot handle all possible policies explicitly, so it needs a mechanism to provide only useful policies from *S*. The POMDP can solve for an optimal policy from the enormous set of available policies, but it requires cost information for the resources. The key insight is that the LP can, via dual costs, provide marginal costs for the resources. Given the current estimates of the dual costs from the LP, the POMDPs can provide new policies for the objects. The algorithm terminates when no policies can be generated that improve the LP solution.

In Chapter II, we develop this decomposition algorithm and prove the algorithm is both convergent and finite. In Chapter III, we survey available POMDP solution methods and their characteristics, and develop the solution procedure used for the sensor-shooter problem. We devote Chapter IV to the actual implementation of the decomposition. We suggest the reader pay particular attention to the computational advice given in this chapter. For implementations requiring runtimes in minutes as opposed to hours, the advice we offer can make the difference between a useable and an intolerable product.

Chapter V addresses the stochastic nature of the general problem and discusses bounds and performance in a stochastic environment. We also discuss the implications of using expected values and continuous variables, and show that the suggested LP provides an upper bound on a "rigid" stochastic problem under certain conditions. We also present simulation results for the sensor-shooter problem, and show that the upper bounds are relatively tight.

Finally, we summarize the results in the dissertation in Chapter VI, and suggest areas for future research.

E. NOTATION

We represent sets with italicized capital or Greek letters. If A and B are sets, then $A+B$ is the union of A and B, and $A - B$ is the set of all elements that are in A but not in B. The symbol $\varnothing$ designates the empty set. The notation $A \subseteq B$ means A is a subset of B, and $|A|$ denotes the number of elements in A. We use braces {} to define the members of a set A, either by listing them, i.e., $A = \{a, b, c, d\}$, or defining a logical condition for inclusion, i.e., $A = \{x: x > 2 \text{ and integral}\}$. In the latter case, the colon should be read as meaning "such that." If a is a member of A, we write $a \in A$; if not we write $a \notin A$. The notation $\langle A, B \rangle$ indicates a "tuple" of sets. If a set is ordered by an index, we may also designate that set as a vector and use the notation $A = (a_i)$. In addition, the statements $A = \{a\}$ or "$a \in A$" mean that a indexes the members of A. The notation $\forall\ a \in A$

means "for all members of set A, indexed by a," and any statement of the form $x \equiv y$ means either x is defined as y or vice versa, depending on context.

We use $\Pr(A)$ to denote the probability of event A, and $\Pr(A|B)$ to denote the probability of event A given event B has occurred. We represent random variables with italicized capital or Greek letters. If X is a random variable, then $\mathrm{E}(X)$ and $\mathrm{Var}(X)$ are the expected value and variance, respectively, of X.

In function definitions, we use the semicolon to separate variables and parameters. For example, the function $u(S;\lambda)$ is a function of a variable S and a set of fixed parameters λ. The notation $x = \arg\max_k \{\alpha^k \cdot \pi\}$ means that x is assigned the index k that maximizes the expression inside the braces; arg min is defined similarly. If α^k and π are vectors, then "$\cdot$" denotes the inner (dot) product of the two vectors.

Theorems, lemmas, and corollaries are numbered consecutively in each chapter, and we designate the conclusion of each proof with the symbol ■. If we introduce a result by citing a source in parentheses, i.e., **Theorem 3.1 (Sondik)**, we are giving that source credit for the proof, but are rewriting it to match our development. Otherwise, we cite the source for the proof explicitly.

In this dissertation, we are combining two disparate bodies of literature, and unfortunately we must define many symbols to describe the theory. We have attempted to use notation common in the literature, but in some cases (particularly for POMDPs), we have modified the notation to fit our needs.

II. THEORETICAL DEVELOPMENT OF THE DECOMPOSITION ALGORITHM

Equations (1.2) and (1.3) present a linear programming representation of the general problem. In this chapter, we develop this LP for a single object, and present a finite decomposition algorithm for its solution. Then, we introduce the POMDP and show it solves the column-generation problem for the single-object LP. We conclude by expanding the problem to consider multiple object types.

A. THE BASIC ALGORITHM FOR A SINGLE OBJECT

1. Notation

For convenience, the following lists the notation used in the remainder of the dissertation for the "master" LP and related subproblems.

- **Sets and Set Indices**

$i \in I$	resources
$j \in J$	object classes
$s \in S_j$	policies for an object of class j
$e \in E$	object states
t	index for time periods 1,2, ..., T
n	index for stages (time periods remaining) 0,1, ... , T

- **Random Parameters**

Y_{sji}	random resources of type i consumed by policy s when applied to an object of type j
R_{sj}	random reward gained for applying policy s to an object of type j

- **LP Parameters**

π_{sje}	probability that an object of type j ends in state e after applying policy s
r_{je}	reward for an object of type j in state e at the end of the time horizon
$E(R_{sj})$	expected reward gained for applying policy s to an object of type j
$E(Y_{sji})$	expected resources of type i consumed by policy s when applied to an object of type j
b_i	nonnegative amount of resource i available
N_j	number of objects of type j available

- **Decision Variables**

x_{sj} the expected number of objects of type *j* controlled by policy *s*

2. The Single-Object LP and the Decomposition Algorithm

The first step in the development of the algorithm is to consider an LP for a single object that can still consume multiple resources over a finite time horizon. We assume the expected rewards $E(R_s)$ and the expected consumptions $E(Y_{si})$ are available as data for all policies $s \in S$. Since there is only one object, we omit the *j* subscripts. Denote this problem as LP(*S*):

$$\text{LP}(S):\quad \max_x \sum_{s\in S} E(R_s)x_s$$

$$\text{st}\quad \sum_{s\in S} E(Y_{si})x_s \le b_i \quad \forall\, i \in I \tag{2.1}$$

$$\sum_{s\in S} x_s = 1 \tag{2.2}$$

$$x_s \ge 0 \quad \forall s \in S.$$

LP(*S*) maximizes the expected reward, subject to constraints on expected resource consumption and expected allocations of policies to the object. Note that $x = \{x_s\}$ is a probability distribution and must sum to 1; also, each x_s is the *expected* number of times policy *s* is applied to the object, as defined above. We assume *S* contains a "null" policy that consumes no resources and has an expected reward of 0, so LP(*S*) is feasible in all cases.

Since $|S|$ is enormous, we cannot enumerate the policies and solve LP(*S*) directly. However, the structure of LP(*S*) implicitly limits the number of columns that can be in any solution. LP(*S*) contains $|I|+1$ constraints. From the basic theory of linear programming, we know that if LP(*S*) has an optimal solution, then it has an optimal basic feasible solution (e.g., Bazarra, Jarvis, and Sherali, 1990, p. 92). Now, a basic feasible solution for LP(*S*) is any solution to the system below, where the inequality constraints are converted to equalities by using the nonnegative slack variables k_i:

$$\begin{aligned} &\sum_{s\in S} E(Y_{si})x_s + k_i = b_i \quad \forall\, i \in I \\ &\sum_{s\in S} x_s = 1 \\ &x_s \geq 0 \quad \forall s \in S \\ &k_i \geq 0 \quad \forall\, i \in I. \end{aligned} \tag{2.3}$$

Any basic solution to the system (2.3) has at most $|I|+1$ nonzero variables, so any basic solution uses at most $|I|+1$ policies (columns) from S (e.g., Bazarra, Jarvis, and Sherali, 1990, pp. 53-54). Since we assume $|I|$ is much smaller than $|S|$, our problem is reduced to finding the appropriate subset of columns from S.

Let v(S) be the value of the optimal solution of LP(S). A lower bound on v(S) is readily available by solving LP(T), where T $\subseteq$ S. To find an upper bound, consider the following Langrangian relaxation (e.g., Parker and Rardin 1988, pp. 206-210) of LP(S), where $\lambda = (\lambda_1, \ldots, \lambda_{|I|})$ is a set of nonnegative parameters:

$$\begin{aligned} \text{LPU(S;}\lambda\text{):}\quad & \max_x \sum_{s\in S} E(R_s)x_s + \sum_{i\in I} \lambda_i \left(b_i - \sum_{s\in S} E(Y_{si})x_s \right) \\ \text{st}\quad & \sum_{s\in S} x_s = 1 \\ & x_s \geq 0 \quad \forall s \in S. \end{aligned} \tag{2.4}$$

Let $u(S;\lambda)$ be the optimal value of LPU($S;\lambda$). Then, we can state the following:

Theorem 2.1 (e.g., Fisher 1985, p. 11): $u(S;\lambda) \geq v(S)$ for any $\lambda \geq 0$.

Proof: Consider a version of LP(S) with the objective function of LPU($S;\lambda$). This new LP differs from LP(S) only in that it has additional nonnegative terms in the objective function, so its optimal value must be at least as large as $v(S)$. Then, remove the constraints from this LP, yielding LPU(S;λ). Since removing constraints cannot decrease the value of the objective, $u(S;\lambda) \geq v(S)$. ■

Furthermore, LPU(S;λ) can be solved without using linear programming. By inspection, the policy in S that maximizes the following gives the optimal solution:

$$u(S;\lambda) = \sum_{i\in I} \lambda_i b_i + \max_{s\in S} \left\{ E(R_s) - \sum_{i\in I} \lambda_i E(Y_{si}) \right\} \qquad (2.5)$$

However, there are two issues in solving (2.5). First, S is very large, implying that (2.5) is difficult to solve. Second, we have no method that specifies the λ's. The first problem is treated in the next section, so assume for the moment we can solve (2.5).

For the second problem, define λ as the vector of dual variables of the resource constraints (2.1) in LP(S). Also, let w denote the dual variable of the allocation constraint (2.2). Assume we are solving this LP via the simplex method. In any particular iteration of the simplex algorithm, each column has a "reduced cost," which can be interpreted as the predicted rate of change in the LP objective value for a unit increase in the variable associated with a particular column. A policy s that is not currently basic (used in the solution at a positive level) can potentially improve the solution if its reduced cost is positive, that is,

$$E(R_s) - \sum_{i\in I} \lambda_i E(Y_{si}) - w > 0 \Rightarrow \text{ possible improvement}. \qquad (2.6)$$

The maximization problem (2.5), which we assume we can solve, finds the policy with the largest reduced cost, and provides a way to find improving columns from S given the current dual values. In essence, (2.5) solves the simplex pricing problem as an external optimization.

The insight above was originally provided in two seminal papers by Gilmore and Gomory (1961, 1963), and is the basis for what is now known as *dynamic column generation*. In problems with a huge number of potential columns, the simplex pricing procedure can often be solved as a separate problem, using the dual variable values from a "master" LP solved with a subset of the available columns. If the subproblem procedure is optimal, it finds an improving column if one exists. Otherwise, there are no more improving columns, and the master problem is optimal.

We can also state the following lemma, which shows that the upper bound and the lower bound are equal at optimality (that is, there is no "duality gap"):

Lemma 2.2: If λ^* is the dual vector of the constraints (2.1) associated with the optimal solution of LP(*S*), then $u(S;\lambda^*) = v(S)$.

Proof: By the complementary slackness theorem of linear programming (e.g., Bazarra, Jarvis, and Sherali 1990, pp. 253-254), at optimality (with solution $x^* = \{x_s^*\}$) the following relationship holds for the constraints (2.1):

$$\lambda_i^*\left[\sum_{s\in S} E(Y_{si})x_s^* - b_i\right] = 0 \quad \forall\, i \in I. \tag{2.7}$$

The solution x^* is feasible for LPU($S;\lambda^*$), and makes its objective function identical to that of LP(*S*). The result follows. ■

With these results, we can a propose dynamic column generation algorithm:

1. **Set $k = 1$, and choose an initial subset T^1 of S.**
2. **Solve LP(T^k) for the initial dual prices λ^k and $v(T^k)$.**
3. **Solve (2.4) using λ^k for $u(S;\lambda^k)$ and policy s.**
4. **If $u(S;\lambda^k) = v(T^k)$ stop; otherwise, let $T^{k+1} = T^k + \{s\}$, add 1 to k and go to 2.**

Theorem 2.3: The dynamic column generation algorithm is finite and converges in at most $|S|$ steps.

Proof: By the basic theory of linear programming, each solution of LP(T^k) is finite. If $s \in T^k$, then $u(S;\lambda^k) = u(T^k;\lambda^k)$, since each upper bound is maximized using policy s. However,

$u(T^k; \lambda^k) = v(T^k)$ by Lemma 2.2, so $u(S; \lambda^k) = v(T^k)$ if $s \in T^k$, and the algorithm converges. If $s \notin T^k$, the algorithm adds a new column. Since there are only $|S|$ policies, at most $|S|$ columns can be added, and the algorithm stops after at most $|S|$ steps. ■

We note that a condition known as *degeneracy* can cause the simplex method to cycle through a sequence of alternate basic representations of the same solution, due to some set of basic variables being in the basic solution at their lower bounds (e.g., Bazaraa, Jarvis, and Sherali 1990, pp. 164-175). We assume that the solution procedure used for the master LP employs methods to prevent cycling, so the algorithm above is still finite.

While the algorithm above is valid, we modify it to improve its computational performance. First, we settle for a solution within some specified Ξ of the optimum, rather than solving to complete optimality. As shown in Chapter IV, this avoids generating a huge number of policies from S, many of which provide insignificant improvement.

Second, we can improve the stopping criterion. One way to implement Ξ-optimal algorithm is to stop when the difference between the upper and lower bounds is smaller than Ξ. Since we are adding potentially improving columns to an LP, the sequence of lower bounds is nondecreasing. However, the same is not true for the sequence of upper bounds (e.g., Bazarra, Jarvis, and Sherali 1990, pp. 321-326). Nonetheless, the minimum of all upper bounds generated is still an upper bound, so we can strengthen the stopping criterion in step 4. The complete Ξ-optimal algorithm is shown in Figure 2.1, and Theorem 2.3 holds for it as well.

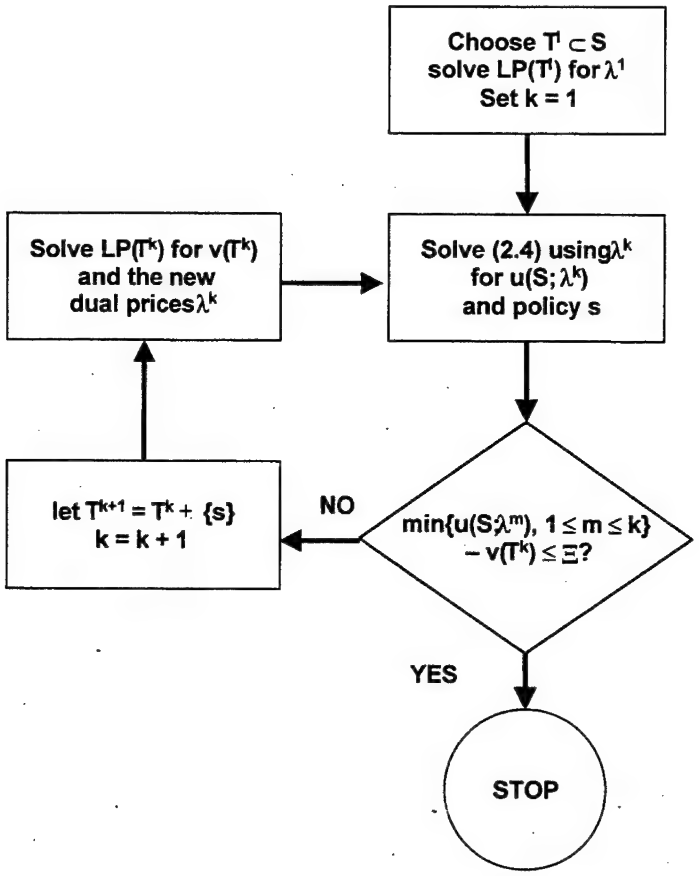
Choose $T^1 \subset S$
solve LP(T^1) for λ^1
Set k = 1
Solve LP(T^k) for v(T^k)
and the new
dual prices λ^k
Solve (2.4) using λ^k
for u(S; λ^k)
and policy s
let $T^{k+1} = T^k + \{s\}$
k = k + 1
NO
min{u(S;λ^m), $1 \le m \le k$}
– v(T^k) $\le \Xi$?
YES
STOP

Figure 2.1: Ξ-Optimal Dynamic Column Generation Algorithm to solve LP(*S*). This algorithm solves the problem of finding the optimal policy to control a single object across time with constrained resources.

B. INTRODUCTION TO THE POMDP SUBPROBLEM

Chapter I gave a very brief introduction to the MDP and POMDP models. The purpose of this section is to give a short history of the POMDP literature, describe the MDP and its extension to a POMDP, and show that solving the POMDP solves the pricing problem (2.6) necessary for the decomposition algorithm. The development of the MDP and the POMDP is from Derman (1970, Chapters 1 and 2), Bertsekas (1976, Chapter 4), and Cassandra (1994, pp. 6-28).

1. POMDP History

POMDPs have a fairly long history in the operations research literature. Drake (1962) is generally credited with the first formulation of a POMDP, but the model was formalized by Sondik (1971), who also offered the first solution algorithms for POMDPs (these algorithms were later shown to have substantial problems, but that is a subject for Chapter III). Subsequent researchers, such as Monahan (1980), Eagle (1984), Cheng (1988), Mukerjee and Seth (1991), and Lovejoy (1991) concentrate on finding improved solution procedures for POMDPs.

However, POMDPs have not enjoyed much actual use. Extensive surveys of applications of Markov decision processes (e.g, White, 1985, 1988, and 1993) document no examples of POMDPs being used in implemented decision support system. Lane (1989) uses real data to *propose* a POMDP to model commercial fishing decisions, but there is no evidence that this work was adopted. Nevertheless, Lane is quite blunt about the lack of applications, stating flatly that "...POMDP models in the literature primarily are contrived problems for the purposes of illustration" (Lane 1989, p. 240). There have been no articles on POMDPs in the mainstream OR journals published in the US since 1991, other than papers by White and Scherer (1994) and Eagle and Thomas (1995). These papers were originally submitted in 1989 and 1987, respectively, so they cannot be regarded as recent work.

However, the artificial intelligence community has embraced the POMDP as a useful tool for robotics applications. Recent papers by Littmann (1994), Cassandra, Littman, and Zhang (1997), and Hauskrecht (1997, 1998) are largely motivated by the need to determine control policies for robots with imperfect sensors. These recent papers have introduced several new algorithms for the solving POMDPs, but have attracted little interest in the OR community.

The lack of popularity of POMDPs is probably due to several factors. First, the POMDP model is concerned with control of a single object with a known, fixed, cost structure for actions, which drastically limits opportunities for applications. Second, POMDPs are difficult to solve, as we discuss in the next chapter; in particular, finding stationary (time-independent) solutions for POMDPs is a very hard problem, and stationary solutions are the desired result for many proposed POMDP applications. As Lovejoy (1991, p. 47) notes, "The significant applied potential for such processes [POMDPs] remains largely unrealized, due to an historical lack of tractable solution methodologies." Puterman (1994, p. 579) uses the latter quote to explain why he omitted POMDPs from his 649-page textbook on Markov decision processes. Finally, as noted by Cassandra (1994, p. 45) the POMDP literature is difficult, and many authors have presented their solution algorithms in a sketchy or incomplete manner. While this is typical of foundation work in all areas of science, the POMDP does not appear to have crossed into the mainstream and gained a generally accepted notation, development, and set of solution methods.

2. The MDP Model

As opposed to the POMDP, the Markov decision process (MDP) model is presented in virtually every introductory OR text (e.g., Hillier and Lieberman 1986, Ross 1993), and is a mainstream technique. MDP'*s* were introduced by Howard (1960) and have a large number of published applications (e.g., White 1993). Therefore, we first present the MDP model and then develop the POMDP from it.

As mentioned in Chapter I, an MDP is specified by a 4-tuple $\langle E, A, P, R\rangle$. As before, E is the set of object states, and A is a set of actions. Similarly, the "laws of motion" are given by the set $P = \{\Pr(e'|a,e)\colon a \in A,\ e \in E,\ e' \in E\}$; each element is the probability that the object transitions from state e to state e' after executing action a (the probabilities in P can also depend on the time period, but we use time-invariant probabilities in this dissertation). Finally, R represents the costs and rewards, and is a set containing the elements $\{r_{ea}, r_e\colon a \in A,\ e \in E\}$.

These entries give the cost incurred or reward gained when the object is in state e and action a is taken, and the terminal reward or cost incurred when the object ends in state e at the end of a finite time horizon. The costs may be deterministic or random; in the latter case, we use expected costs. Note also that the rewards are related to the time period through the actions.

Let E_t be the state of the object at time t, and A_t be the action taken in time period t; both variables are capitalized to indicate they are random. Furthermore, define the *information vector* $I_t = \{E_n, A_n\colon n = 1,2,\ldots t\}$ as the history of the system up to time t. Suppose we have some rule that we use to determine which action to take in time t based on E_t and I_{t-1}. Given this rule, the specification $\langle E, A, P, R\rangle$, and an initial state, the sequence $\{E_t, A_t, t = 0,1,\ldots\}$ is a stochastic process that is called an MDP. Note that this process is not necessarily a Markov process; the decision rule may use the entire history, so the process may not satisfy the Markov property. Nonetheless, all such processes are called MDP'*s*.

3. MDP Policies and Solutions

The description of an MDP does not specify what it means to solve such a process. At this point, we have a set of rewards and a system model, and it seems reasonable that a solution involves finding a rule for taking actions that maximizes some form of the reward. In this section, we formally define these rules and the form of the reward we are interested in.

We first discuss the rules, which we call policies. The following definition is the most general one, and formalizes the one given in Section I.C.1:

Definition 2.1: A **policy** s for an MDP is a map from the history of the process and the current state to a set of actions for each time period. Formally, s is an *admissible* policy if and only if

$$s = \{s_a(I_{t-1}, e_t), a \in A, \ t = 1,2,\ldots\}, \text{ such that}$$
$$0 \le s_a(I_{t-1}, e_t) \le 1, \quad \sum_{a \in A} s_a(I_{t-1}, e_t) = 1 \quad \forall \, I_{t-1}, e_t. \tag{2.8}$$

Here $s_a(I_{t-1}, e_t)$ is the probability of taking action a for the information vector I_{t-1} and the state e_t. Set Δ denotes the set of all admissible policies.

Definition 2.1 allows for the choice of action to be randomized based on the history and current state. While this definition allows for very general policies, it also means that $|\Delta|$ is infinite, making the idea of searching Δ for the best policy daunting.

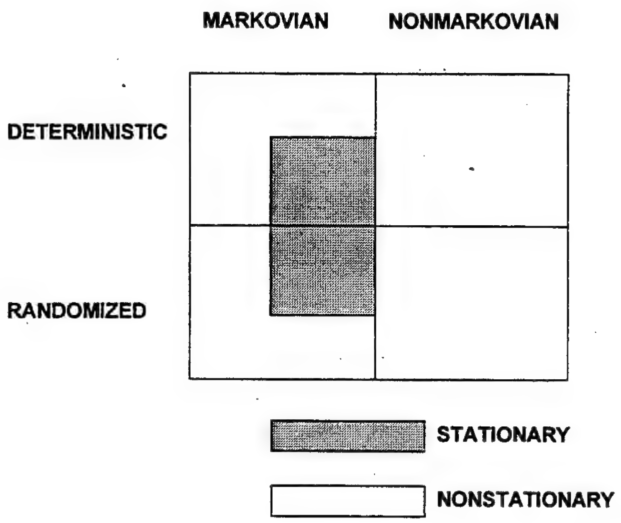

Figure 2.2: Classification of MDP Policies. In this dissertation, we concentrate on models that require only nonrandomized, Markovian, nonstationary policies.

However, Δ has manageable subsets. Let Δ_m be the subset of *Markovian* policies, where the action taken depends only on the current state and the time period t. Let Δ_s be the subset of *stationary* policies, which only depend on the current state and the action; $\Delta_n = \Delta - \Delta_s$ denotes the subset of *nonstationary* policies. Δ_d is the subset of *deterministic* (non-randomized) policies. In many references (e.g., Derman 1970, pp. 6-7), the deterministic policies are a subset of the stationary policies. Here, we allow intersections of the primary subsets, so that Δ_{mnd} is the set of Markovian, nonstationary, deterministic policies. Figure 2.2 depicts this classification scheme.

With the notion of a policy defined, we must specify what we want to optimize, as there are several possible problems. In most textbooks (e.g., Ross 1993) the emphasis is on finding the total expected reward over an infinite time horizon, which requires discounting rewards across time to yield a finite expected reward. Another commonly-addressed problem is that of finding the maximum expected reward per unit time, which again is concerned with infinite horizons. A third problem is to maximize the expected reward prior to reaching some "stopping" state, where the process ceases.

However, our aim is simpler: we want to maximize the expected reward attained at the end of a finite time horizon. In particular, we address MDP's where an object can be controlled across a time horizon with some allowable set of actions. We receive a terminal reward r_e if the object ends in state e. However, each action consumes a random vector of resources ($W_i(e,a)$), and the resources have fixed per-unit costs given by the vector $\lambda = (\lambda_i)$. Therefore,

$$r_{ea} \equiv E\left[-\sum_{i\in I} \lambda_i W_i(e,a)\right], \text{ and } R = \{r_{ea}, r_e \colon a \in A,\ e \in E\}.$$

Let T be the number of time periods in the horizon, and let R_s be the random terminal reward gained after following policy s. The objective is to find a policy that maximizes the expected reward given the starting state e_1 and the costs λ:

$$\begin{aligned} val_T(e_1,\lambda) &= \max_{s\in\Delta} E\left[R_s - \sum_{t=1}^{T}\sum_{i\in I}\lambda_i W_i(E_t,A_t)\right] \\ &= \max_{s\in\Delta} E[R_s] - \sum_{t=1}^{T}\sum_{i\in I}\lambda_i E[W_i(E_t,A_t)]. \end{aligned} \tag{2.9}$$

The following theorem and corollary give a solution procedure for this MDP and classify the optimal policies:

Theorem 2.4: Given a *T*-period MDP of the form above, specified by <*E, A, P, R*>, the following dynamic programming recursion maximizes (2.9):

$$\begin{aligned} val_0(e,\lambda) &= r_e, \\ val_n(e,\lambda) &= \max_{a\in A}\left\{-\sum_{i\in I}\lambda_i E[W_i(e,a)] + E[val_{n-1}(e',\lambda)]\right\} \\ &= \max_{a\in A}\left\{-\sum_{i\in I}\lambda_i E[W_i(e,a)] + \sum_{e'\in E} val_{n-1}(e',\lambda)\Pr(e'|a,e)\right\}, \\ & n = 1,2,\ldots T. \end{aligned} \tag{2.10}$$

Corollary 2.5: For any starting state e_1, the optimal policy that takes actions in accordance with the DP recursion (2.10) is Markovian and deterministic; that is $s^* \in \Delta_{md}$.

Proof: see Derman (1970, pp. 12-14). ■

Theorem 2.4 and Corollary 2.5 show that it is not necessary to consider randomized or non-Markovian policies; the optimal policy is always be a member of Δ_{md}. Note that the optimal policy may be stationary or nonstationary.

4. The MDP as a Solution to the Column-Generation Problem

The optimal expected payoff $val_T(e_1,\lambda)$ is computed directly by the recursion (2.10). However, additional work is required to determine the distribution of the terminal object states and compute the expected consumptions (a point generally not made in textbook discussions of MDP'*s*). To get this information, we must compute the expected consumptions and terminal state probabilities during the recursion, or else save the policy in some form and compute the necessary information from it.

If we do compute this information, then we can make the connection between the column-generation problem (2.5) and the solution of the MDP. Returning to the notation introduced in Section II.A.1 (but suppressing the index *j* because we are only controlling a single object), let π_{se} be the probability that the object's terminal state is *e*, given we use policy *s* and start in state e_1. Then $E[R_s] = \sum_{e \in E} r_e \pi_{se}$. Furthermore, the random consumptions are related by the equation $Y_{si} = \sum_{t=1}^{T} W_i(E_t, A_t)$. Letting $S = \Delta_{md}$ leads to the following theorem:

Theorem 2.6: Solving the MDP (2.10) solves the column-generation problem (2.5) **under the conditions of perfect observability.**

Proof: We can write val$_T(e_1,\lambda)$ equivalently as

$$\begin{aligned} val_T(e_1,\lambda) &= \max_{s \in S} E\left[R_s - \sum_{t=1}^{T} \sum_{i \in I} \lambda_i W_i(E_t, A_t) \right] \\ &= \max_{s \in S} \left\{ E[R_s] - E\left[\sum_{i \in I} \lambda_i Y_{si} \right] \right\} \\ &= \max_{s \in S} \left\{ E(R_s) - \sum_{i \in I} \lambda_i E(Y_{si}) \right\}. \end{aligned}$$

From (2.5), $u(S;\lambda) = \sum_{i \in I} \lambda_i b_i + \max_{s \in S} \left\{ E(R_s) - \sum_{i \in I} \lambda_i E(Y_{si}) \right\}$, so

$$val_T(e_1,\lambda) = u(S;\lambda) - \sum_{i \in I} \lambda_i b_i . \tag{2.11}$$

The optimal MDP value function val$_T(e_1,\lambda)$ only differs from $u(S; \lambda)$ by a constant. Therefore, the policy determined by the DP recursion (2.10) also solves (2.5). ■

This appears to be an original result, as we have not found any papers in the course of this research that solve MDPs as subproblems in a decomposition algorithm. The key to the decomposition is recognizing that the MDP recursion (2.10) is equivalent to solving the simplex pricing problem (2.5), which requires rewriting the MDP solution value in the form of (2.11). A problem with complete observability could be solved using a master LP and an MDP subproblem; for example, if all sensors were perfect in the sensor-shooter problem, the subproblem could be modeled as an MDP. Indeed, any problem where perfect information is available but constrained that meets the other conditions of Section I.B could be solved using the Ξ-optimal algorithm of Figure 2.1.

However, we must address the issue of partial observability. Our observations are subject to error, so we are uncertain about the actual states of the objects. Fortunately, theory exists for this case as well.

5. Extension to the POMDP Model

To account for partial observability, we must choose how to make a decision without knowing the precise state of the object. In the POMDP, the sets E, A, P and R are identical to those in the MDP. However, the POMDP also includes a finite set Θ of possible observation outcomes, and the set $B = \{\Pr(\theta|a,e) : \theta \in \Theta, a \in A, e \in E\}$, where each $\Pr(\theta|a,e)$ is the probability of observing θ after taking action a and the object is in state e. We assume without loss of generality that the probabilities in B are time-invariant.

The set B also defines the allowable combinations of actions and observations in the POMDP. Virtually all example POMDPs in the literature assume that all combinations of actions and observations are possible, and this may lead a reader to believe that such an assumption is a requirement of the model. However, the standard POMDP specification above can restrict the action-observation combinations via the probabilities in B. If an observation θ cannot occur after taking action a, then $\Pr(\theta|a,e) = 0$ for all states e. The ability to restrict observations is important in the sensor-shooter problem, because an attacking aircraft may be incapable of observing the state of the target after the attack; similarly, a particular sensor may only provide certain types of observations, such as imagery.

We also need to specify the order of events in a time period. Our convention in this dissertation is that within a time period, the action occurs first, then the object transition, then the observation, then we incur the costs of the action. So, an object begins in state e_1, we take action a_1, the object transitions to state e_2, and the observation θ_1 and the cost incurred is based on the state e_2. No actions can be taken using information gained from observation θ_1 until the next period. Figure 2.3 depicts the dynamics of this process.

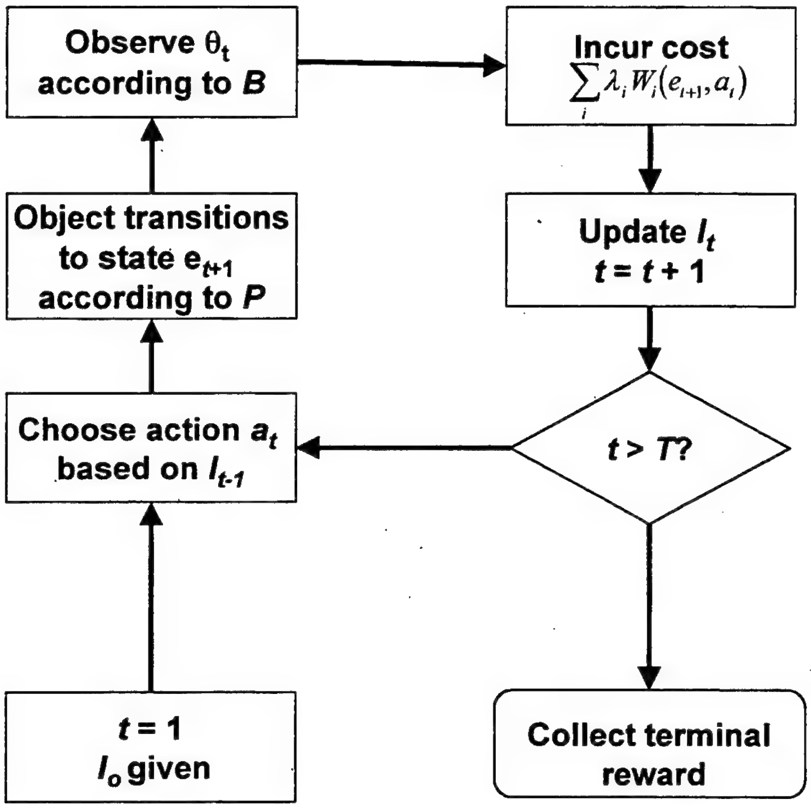
Observe θ_t according to B
Incur cost $\sum_i \lambda_i W_i(e_{t+1}, a_t)$
Object transitions to state e_{t+1} according to P
Update I_t $t = t + 1$
Choose action a_t based on I_{t-1}
$t > T$?
$t = 1$ I_0 given
Collect terminal reward

Figure 2.3: Dynamics of the Finite-Horizon POMDP. Given an initial history I_0, the POMDP successively chooses actions and collects observations until it terminates. The actions taken in each period have random costs, and the terminal reward depends on the ending state of the object. We use the history $I_t = \{\theta_n, A_n, n = 1, 2, \ldots, t\}$ to choose the action.

The POMDP model also requires revision of the notions of a history and a policy. The availability of an action can no longer depend on the state of an object, because we cannot observe the state with certainty. However, we do have access to the history of actions (A_t) and observations (θ_t), so we can define the POMDP history as the information vector

$I_t = \{\theta_n, A_n, n = 1,2,\ldots t\}$. Note that we do not include the costs incurred in each stage in the information vector; if they depend on the true state of the object and are also partially observable, we can determine them implicitly using θ_t. As for policies, the analog of Definition 2.1 for POMDPs is as follows:

Definition 2.2: A **policy** *s* for a POMDP is a map from the history of the process to a set of actions for each time period. A policy is admissible if and only if

$$\begin{aligned} &s = \{s_a(I_{t-1}),\ a \in A,\ t = 1,2,\ldots\}, \text{ such that} \\ &\quad 0 \le s_a(I_{t-1}) \le 1, \quad \sum_{a \in A} s_a(I_{t-1}) = 1, \quad t = 1,2,\ldots \end{aligned} \tag{2.12}$$

As before, Δ is the set of all admissible policies.

With policies defined this way, it is possible to use the MDP to solve this problem. The set of all possible histories $\{I_t\}$ is perfectly observable and finite for all *t*, and it can serve as the state space. This converts the problem to an MDP. .

Unfortunately, this MDP is untenable for all but the smallest of problems. In each time period there can be up to $(|A||\Theta|)^{t-1}$ possible histories, and we would have to evaluate each of these in a DP recursion. Consider an instance of the sensor-shooter problem. A target has 2 states (live or dead), and the actions would normally include about 50 attack tactics and 10 different observation sensors. If we assume the sensor information is translated into a direct judgement on the target's state, then there are 2 observation outcomes for each sensor look. If we assume the attackers cannot assess the target, then they can only provide a "null" observation. As a result, the set of histories for period 10 would contain $(50 \times 1 + 10 \times 2)^9 = 4.04 \times 10^{16}$ states, and finding the optimal policy for all such states (as well as for the prior time periods) would involve a prohibitive amount of computing. The conclusion is that $\{I_t\}$ is a legitimate but computationally unmanageable state space for the partially observable problem.

However, if it were possible to find a quantity that represented all the information in each state I_t more compactly, this quantity could be used as the state. The notion of representing I_t by such a quantity (known as a *sufficient statistic*), was popularized for stochastic control in a paper by Striebel (1965). Sondik (1971, pp. 16-17) proposed a particular sufficient statistic for POMDPs that makes both mathematical and functional sense. The idea is straightforward. Since we know the probabilities in the sets P and B, and we know the sequence of actions and observations, we can compute the probability that the object is in state e, given the history I_t. Suppose we are executing a policy s and are at some particular stage. Define the vector $\pi = \left(\pi_e : \sum_{e \in E} \pi_e = 1 \right)$ to be the current *belief state*, or the probability distribution over the states. Suppose we take action a in a particular time period and then observe θ. Then, the revised belief state probabilities (π_e') can be computed via Bayes' Theorem:

$$\begin{aligned} \pi'_e &= \frac{\Pr(\theta|a,e)\Pr(e|\pi,a)}{\Pr(\theta|\pi,a)} \\ &= \frac{\Pr(\theta|a,e)\sum_{e'\in E}\pi_{e'}\Pr(e|a,e')}{\sum_{e''\in E}\left[\Pr(\theta|a,e'')\sum_{e'\in E}\pi_{e'}\Pr(e''|a,e')\right]} \quad \forall e \in E. \end{aligned} \tag{2.13}$$

The revision formula (2.13) applies for actions that can both affect the state of the object and provide observations. If the action only affects the state and does not provide an observation (for example, attacking a target with a long-range weapon with no accompanying sensors), then there is only a single "null" observation possible, and it occurs with probability 1. This reduces (2.13) to

$$\pi'_e = \sum_{e'\in E}\pi_{e'}\Pr(e|a,e') \quad \forall e \in E. \tag{2.14}$$

Similarly, for actions that only observe (such as imaging a target with as satellite), the object cannot pass to another state. Therefore, Pr(e| a,e) = 1, in which case

$$\pi'_e = \frac{\Pr(\theta|a,e)\pi_e}{\sum_{e'\in E}\pi_{e'}\Pr(\theta|a,e')} \quad \forall e\in E. \tag{2.15}$$

In all cases, the revision formulas (2.13), (2.14), and (2.15) provide Bayesian updates to the current belief state. In addition, these formulas can be applied recursively, so there is no need to explicitly maintain the history I_t. The belief state vector for period t, denoted $\pi(t)$, represents all the information available in I_t.

Theorem 2.7: $\pi(t)$ is a sufficient statistic for I_t.

Proof: see Bertsekas (1976, pp. 122-126). ■

Using this sufficient statistic simplifies the problem and maintains perfect observability. Consequently, we can transform the partially observable problem to a perfectly observable MDP. To avoid the combinatorial growth of the $\{I_t\}$ sets, we only need to maintain the π vector from period to period.

Let $Tr(\pi|\ a,\ \theta)$ be the transformation function for (2.13); that is, $\pi(t+1) = Tr[\pi(t)|a,\theta]$. Then, a DP recursion for the POMDP analogous to (2.10) is

$$\begin{aligned} val_0[\pi,\lambda] &= \sum_{e\in E}\pi_e r_e, \\ val_n[\pi,\lambda] &= \max_a\left\{-\sum_{i\in I}\lambda_i\left(\sum_{e\in E}\pi_e E[W_i(e,a)]\right) + E\left[val_{n-1}\left(Tr[\pi|a,\theta],\lambda\right)\right]\right\}, \\ n &= 1,2,\ldots T. \end{aligned} \tag{2.16}$$

However, using the sufficient statistic $\pi(t)$ as the state transforms the state space from the set of histories, which is finite, to the set of all probability distributions over all the object states. This set, which we denote Π, is noncountably infinite, and a policy for a POMDP using $\pi(t)$ as a belief state is now a map from a noncountably infinite set to the set of actions. This means the set Δ^{Π} of policies is also noncountably infinite. Consequently, we may be tempted to employ (2.16) directly without asking whether DP can still find the optimal policy. Derman (1970, p. 16) points out that Theorem 2.4 does not necessarily hold for such state spaces, so (2.16) may not be valid.

Fortunately, this is not the case:

Theorem 2.8: The sequence $\{\pi(t), \theta_t, A_t\}$ of belief states, observations, and actions is a Markov process, and the DP recursion (2.16) finds optimal policy s^*. Furthermore, $s^* \in \Delta^{\Pi}_{md}$.

Proof: see Sondik (1971, pp. 16-25). ■

Therefore, if we set $S = \Delta^{\Pi}_{md}$ and denote the initial belief state as $\pi(1)$, the recursion (2.16) has the same relationship to the column-generation problem (2.5) as did the MDP case, that is,

$$val_T[\pi(1), \lambda] = u(S; \lambda) - \sum_{i \in I} \lambda_i b_i . \qquad (2.17)$$

In other words, Theorem 2.6 also applies to the solution of the POMDP using the sufficient statistic $\pi(t)$ as a state. **This is the fundamental idea exploited in this dissertation: solving a POMDP for the optimal policy also solves the column-generation problem in the decomposition algorithm.** The recursion (2.16) implicitly searches all candidate polices in S, as required by the ε-optimal algorithm. The POMDP subproblem provides columns for the current set of resource prices. It also provides an upper bound in the overall decomposition because it in itself is an optimization.

We are "overloading" the definition of a policy in this development, which may be particularly confusing for readers conversant in MDPs and POMDPs. A policy is a sequence of maps from the state to the set of actions for MDPs and POMDPs. However, we are also using the term policy to describe an expected reward and a vector of expected resource consumptions as a column in a linear program. We do not define notation to divorce the POMDP and LP usage, because the relation between the two is the foundation of this dissertation.

There is a critical issue remaining with the POMDP subproblem. We call $\Pi = \left\{ \pi : \sum_{e \in E} \pi_e = 1 \right\}$ the *belief space*, and use it as the state space for the POMDP. As we noted, Π is the set of all probability distributions over the states in E, and using it instead of the set of all possible histories means that we have switched from using a finite state space to using an *uncountably infinite* state space. It is true that we only have to update the vector π rather than iterating over all possible histories, but we also have to compute optimal actions for an uncountably infinite number of belief states. We can write the POMDP recursion (2.16) in a compact form, but we cannot solve it using the enumeration methods we could use for an MDP.

However, it seems reasonable that a finite algorithm should exist to solve the POMDP even using the belief space. After all, the set of histories is finite, and we have already discussed a finite (but combinatorially intractable) method of using the history as the state. As it turns out, the value functions for (2.16) have a particular structure we can exploit, and we devote Chapter III to the discussion of the characteristics of (2.16) and the available solution algorithms.

So far, we have demonstrated that a finite algorithm exists using an LP master problem and a POMDP (or MDP) subproblem for control of a single object using multiple constrained resources. This result is useful, but limited; we would like to consider more than one object. We conclude this chapter by addressing this case.

C. GENERALIZING TO MULTIPLE OBJECTS

We now extend the algorithm to the case where we are controlling a number of classes of identical objects. As before, *J* is the set of classes. However, now we define $S = \bigcup_{j \in J} S_j$; that is, the total set of policies is the union of the admissible policies available for each class. Furthermore, we assume each policy in S_j only affects the object in class *j* that it controls. Using the notation defined in Section II.A.1, the multiple-object problem LP2(*S*) is given below, with the dual variables for the constraints in parentheses:

$$\begin{aligned}
\text{LP2}(S)\text{:}\quad & \max_x \sum_{j \in J} \sum_{s \in S_j} E(R_{sj}) x_{sj} \\
\text{st}\quad & \sum_{j \in J} \sum_{s \in S_j} E(Y_{sji}) x_{sj} \le b_i \quad \forall\, i \in I \qquad (\lambda_i) \\
& \sum_{s \in S_j} x_{sj} = N_j \quad \forall\, j \in J \qquad (w_j) \\
& x_{sj} \ge 0 \quad \forall\, j \in J, s \in S_j .
\end{aligned} \tag{2.18}$$

Since we assume the rewards are additive over the objects, the objective function is justified. Also, the policies available to each class only affect single objects, so the expected consumptions are additive as well and the constraints are justified. For feasibility, we again assume that each set S_j contains a null policy that consumes no resources and gives no reward.

The upper bound for this extended problem is similar to the single-object upper bound:

$$\begin{aligned}
\text{LPU2}(S;\lambda)\text{:}\quad & \max_x \sum_{j \in J} \sum_{s \in S_j} E(R_{sj}) x_{sj} + \sum_{i \in I} \lambda_i \left(b_i - \sum_{j \in J} \sum_{s \in S_j} E(Y_{sji}) x_{sj} \right) \\
\text{st}\quad & \sum_{s \in S_j} x_{sj} = N_j \quad \forall\, j \in J \\
& x_{sj} \ge 0 \qquad \forall\, j \in J, s \in S_j .
\end{aligned} \tag{2.19}$$

However, LPU2($S;\lambda$) decomposes into $|J|$ separate optimizations, one for each object class. Define $u_j(S_j;\lambda)$ as

$$u_j\left(S_j;\lambda\right)=\max_{s\in S_j}\left\{E\left(R_{sj}\right)-\sum_{i\in I}\lambda_i E\left(Y_{sji}\right)\right\}\equiv val_T^j\left[\pi^j(1),\lambda\right]. \tag{2.20}$$

Then, the overall upper bound is given by

$$u(S;\lambda)=\sum_{i\in I}\lambda_i b_i+\sum_{j\in J}N_j u_j\left(S_j;\lambda\right). \tag{2.21}$$

From the standpoint of the POMDP subproblems, this algorithm is exactly the same as the single-object algorithm. Each of the |J| POMDPs must be solved to yield $val_T^j\left[\pi^j(1),\lambda\right]$, but these subproblems are independent. The resources are still constrained, but the fact that other objects are competing for them has no effect. The costs of the resources, as before, are communicated via the dual prices. Theorem 2.1 concerning the upper bound holds by simple extension, as does Theorem 2.3 on the convergence and finiteness of the decomposition algorithm.

The resulting multiple-object-class algorithm is shown in Figure 2.4. The only modifications to the original algorithm are the need to find an initial set of policies $T^{j,1}$ for each object class, and the need to solve a POMDP subproblem for each class. The number of objects in each class has *no* effect on the decomposition; indeed, the algorithm is better suited to problems with large numbers of objects and few classes.

One difference between the multiple- and single-object algorithms is that some of the object classes may not generate policies that can improve the current LP solution. There is no guarantee that for all $j\in J$, $u\left(S;\lambda^k\right)-w_j>0$ in any iteration k. In our implementation of the algorithm in Chapter IV, we omit policies that do not price favorably. However, including them cannot decrease the lower bound, and they may price favorably later as the dual variables change. Therefore, we have written the general algorithm to use them automatically.

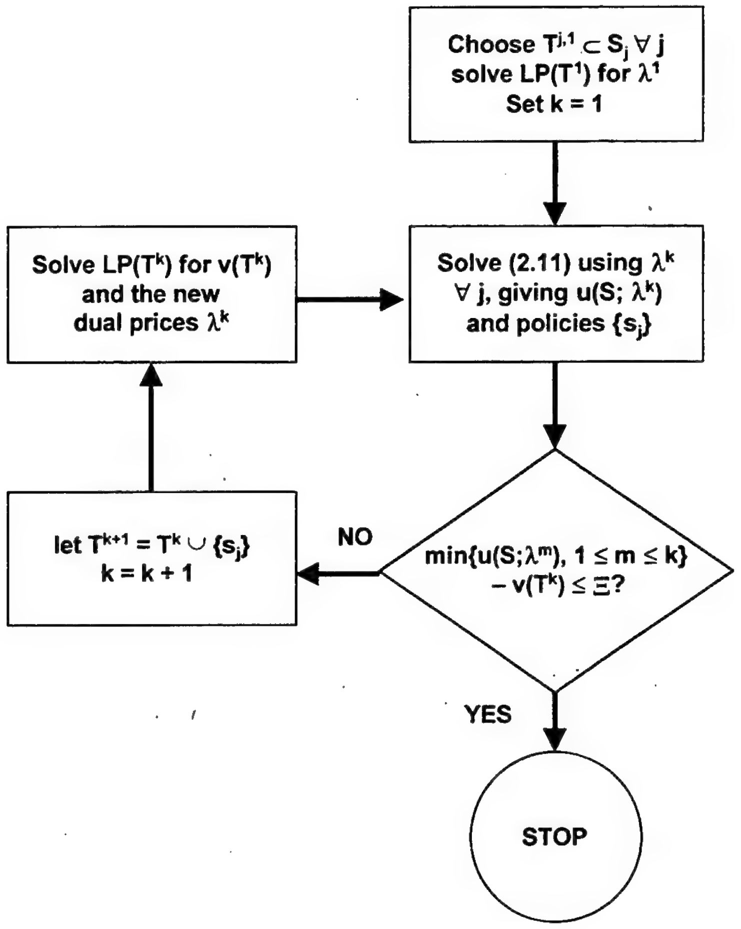
Choose $T^{j,1} \subset S_j \; \forall j$
solve LP(T^1) for λ^1
Set k = 1
Solve (2.11) using λ^k
$\forall$ j, giving u(S; λ^k)
and policies $\{s_j\}$
Solve LP(T^k) for v(T^k)
and the new
dual prices λ^k
$\min\{u(S;\lambda^m), 1 \le m \le k\}$
$- v(T^k) \le \Xi$?
NO
let $T^{k+1} = T^k \cup \{s_j\}$
k = k + 1
YES
STOP

Figure 2.4: Ξ-Optimal Dynamic Column Generation Algorithm for Multiple Object Classes.

This algorithm also applies to multiple-object problems with MDP (perfect information) subproblems. For example, an article by Meuleau et al. (1998) attempts to solve a large-scale MDP with resource constraints. The authors note that a straightforward attempt to solve such an MDP would require that the number and types of objects, as well as the numbers of each resource remaining, be incorporated into the state space. The resulting MDP is intractable, and the authors propose an approximate method for solving such problems. We can solve such problems exactly and also address partial observability, so long as we are allowed to constrain resource consumption on the average. In the paper by Meleau et al., the straightforward MDP constrains resources for all possible outcomes, so no violations of the constraints are possible. Constraining expected resource consumptions may lead to violations, and the consequences of those violations depend on the situation being modeled. We analyze this issue at length in Chapter V.

D. SUMMARY

In this chapter, we have developed a finite decomposition algorithm that solves the problem posed in Chapter I. By using a master LP to provide implicit prices on constrained resources and MDP or POMDP subproblems to provide improving policies, we present a finite dynamic column generation procedure. This algorithm, and the insight that an MDP or a POMDP could be used to solve the column generation problem, is the key idea in this dissertation. Furthermore, the algorithm shown in Figure 2.4 can be solved to any desired level of precision due to the computation of upper and lower bounds.

III. SOLUTION OF THE POMDP SUBPROBLEM

The development in Chapter II stopped with the specification of a DP recursion (2.16) to solve the POMDP. However, we noted that this specification required using an uncountably infinite state space, and that the enumeration techniques used in solving MDP's were not applicable. In this chapter, we cover the known characteristics of the POMDP solution (piecewise linearity and convexity/concavity). Also, we survey the available methods for solving POMDPs and suggest which algorithms might work best in the decomposition. Finally, we discuss the algorithm we employ (the linear support algorithm) in detail.

A. POMDP SOLUTION CHARACTERISTICS

In Sections III.A. and III.B., we focus on solving the POMDP for a single object. As in Section II.B.5, we use $\pi = \left(\pi_e : \sum_{e \in E} \pi_e = 1 \right)$ as the belief state, and $\Pi = \{\pi\}$ as the belief space; that is, Π is the set of all probability distributions on E. However, we suppress λ as a argument for the value function, as we are concentrating on solving a problem with specified resource costs.

1. Piecewise-Linearity and Convexity

With n indexing the stages for a T-stage horizon, the DP recursion for the POMDP is

$$val_0(\pi) = \sum_{e \in E} \pi_e r_e ,$$

$$val_n(\pi) = \max_{a \in A} \left\{ -\sum_{i \in I} \lambda_i \left(\sum_{e \in E} \pi_e E[W_i(e,a)] \right) + E\left[val_{n-1}\left(Tr[\pi \mid a, \theta] \right) \right] \right\}, \quad (3.1)$$

$$n = 1, 2, \ldots T .$$

This recursion resists simple enumeration due to the continuity of π. Nonetheless, we know already that there are a finite number of Markovian, deterministic policies. Therefore, even with this state space there must be a finite way to find and specify the optimal policy. As it turns out, the key insight was discovered by Sondik (1971, pp. 26-28), who proved that $val_n(\pi)$ is piecewise-linear and convex (or concave if the objective is minimization). We present the proof in order to tailor it for the general POMDP used in this dissertation.

Theorem 3.1 (Sondik): *$val_n(\pi)$* is piecewise-linear and convex.

Proof: Sondik's proof uses induction and some algebra (Littman (1994, pp. 15-17) gives an interesting alternative proof using decision trees). By definition, $val_0(\pi)$ is a linear function of π, so it is trivially piecewise-linear and convex. If we can show that $val_n(\pi)$ is the maximum of a finite set of linear functions, then it is piecewise-linear and convex because the maximum of a finite set of linear functions is piecewise-linear and convex.

For this proof, we use the following notation: $P^a \equiv [\Pr(e|a,e')]$ is an |E| x |E| matrix of transition probabilities. $\Theta_\theta^a \equiv diag[\Pr(\theta|a,e)]$ is an |E| x |E| matrix with the observation probabilities on the diagonal. $R_e = (r_e)$ is an |E| x 1 vector of expected terminal rewards, and $R_a = (r_{ea})$ is an |E| x 1 vector of expected action costs. Let *1* be an |E| x 1 vector of 1's. With this notation, we can write the following:

$$\begin{aligned}
&\Pr(\theta|\pi,a) = \pi \cdot P^a \cdot \Theta_\theta^a \cdot 1, \\
&Tr[\pi|a,\theta] = \frac{\pi \cdot P^a \cdot \Theta_\theta^a}{\pi \cdot P^a \cdot \Theta_\theta^a \cdot 1}, \\
&r_{ea} = \sum_{i\in I} \lambda_i E[W_i(e,a)].
\end{aligned} \tag{3.2}$$

We now rewrite the DP recursion (2.16):

$$
\begin{aligned}
&val_0(\pi) = \pi \cdot R_e, \\
&val_n(\pi) = \max_{a\in A}\left\{-\pi \cdot R_a + E\left[val_{n-1}\left(Tr\left[\pi \,|\, a,\theta\right]\right)\right]\right\} \\
&\qquad = \max_{a\in A}\left\{-\pi \cdot R_a + \sum_{\theta\in\Theta} \Pr(\theta | \pi, a)\, val_{n-1}\left(Tr\left[\pi \,|\, a,\theta\right]\right)\right\}, \\
&\qquad n = 1,2,\ldots T.
\end{aligned}
\tag{3.3}
$$

We now show that the value function is piecewise-linear and convex for $n = 1$:

$$
\begin{aligned}
val_1(\pi) &= \max_{a\in A}\left\{-\pi \cdot R_a + E\left[val_{n-1}\left(Tr\left[\pi \,|\, a,\theta\right]\right)\right]\right\} \\
&= \max_{a\in A}\left\{-\pi \cdot R_a + \sum_{\theta\in\Theta} \Pr(\theta | \pi, a)\, val_0\left(Tr\left[\pi \,|\, a,\theta\right]\right)\right\} \\
&= \max_{a\in A}\left\{-\pi \cdot R_a + \sum_{\theta\in\Theta} \pi \cdot P^a \cdot \Theta^a_\theta \cdot 1 \frac{\pi \cdot P^a \cdot \Theta^a_\theta}{\pi \cdot P^a \cdot \Theta^a_\theta \cdot 1} \cdot R_e\right\} \\
&= \max_{a\in A}\left\{\pi \cdot \left(-R_a + \sum_{\theta\in\Theta} P^a \cdot \Theta^a_\theta \cdot R_e\right)\right\}.
\end{aligned}
$$

Letting $\alpha^a \equiv -R_a + \sum_{\theta\in\Theta} P^a \cdot \Theta^a_\theta \cdot R_e$, we can write

$$
val_1(n) = \max_{a\in A}\left\{\pi \cdot \alpha^a\right\}. \tag{3.4}
$$

Since the value function is the maximum of a set of linear functions, it is piecewise-linear and convex for $n = 1$.

At this point, we detour from the proof to discuss the α-notation introduced above. This notation is common in the POMDP literature, and it demonstrates how the POMDP policy is actually stored. In any stage n, we need a map from the history of the POMDP to the actions. The belief state π provides a sufficient statistic for the history, but we also need a way to determine the optimal action given π. To provide this, we store the set $V(n)$ of α-vectors $\alpha^k(n)$ for every stage n, and also store the action a associated each α-vector. Let $act(k,n)$ be a function that returns the action for α-vector k in stage n. For any belief state π, we can then compute the value function $val_n(\pi)$ and the action $a_n(\pi)$ using the following:

$$val_n(\pi) = \max_k\{\pi \cdot \alpha^k(n)\},$$
$$a_n(\pi) = act\left(\arg\max_k\{\pi \cdot \alpha^k(n)\}, n\right). \qquad (3.5)$$

Returning to the proof, we now show the optimal POMDP policy can be written using in the desired form for all stages. Assume $val_{n-1}(\pi) = \max_k\{\pi \cdot \alpha^k(n-1)\}$ for stage n-1. Then

$$val_n(\pi) = \max_{a\in A}\left\{-\pi \cdot R_a + \sum_{\theta\in\Theta} \Pr(\theta|\pi, a)\left[\max_k\{Tr(\pi|a,\theta)\cdot \alpha^k(n-1)\}\right]\right\}.$$
$$= \max_{a\in A}\left\{\begin{matrix}-\pi \cdot R_a + \\ \sum_{\theta\in\Theta} \pi \cdot P^a \cdot \Theta^a_\theta \cdot 1 \ \max_k\left\{\frac{\pi \cdot P^a \cdot \Theta^a_\theta}{\pi \cdot P^a \cdot \Theta^a_\theta \cdot 1}\cdot \alpha^k(n-1)\right\}\end{matrix}\right\}$$
$$= \max_{a\in A}\left\{\sum_{\theta\in\Theta} \max_k\left\{\pi \cdot \left[-\frac{R_a}{|\Theta|} + P^a \cdot \Theta^a_\theta \cdot \alpha^k(n-1)\right]\right\}\right\}. \qquad (3.6)$$

Now, let the notation B^A, where B and A are arbitrary sets, be the set of all functions from A to B. Then $|B^A| = |B|^{|A|}$. Use this notation to define the following set of vectors $VM(n)$:

$$VM(n) = \left\{\begin{matrix}\alpha(n): \alpha(n) = \sum_{\theta\in\Theta}\left[-\frac{R_a}{|\Theta|} + P^a \cdot \Theta^a_\theta \cdot \alpha^{k(\theta)}(n-1)\right], \\ a \in A,\ k(\theta) \in V(n-1)^\Theta\end{matrix}\right\}. \qquad (3.7)$$

This set contains $|V(n-1)|^{|\Theta|}$ vectors for each action a, and therefore contains a total of $|A||V(n-1)|^{|\Theta|}$ vectors. Since $|V(1)| = |A|$ and is finite, $|A||V(n-1)|^{|\Theta|}$ must also be finite. We can now write the stage n value function as

$$val_n(\pi) = \max_{\alpha(n) \in VM(n)} \{\pi \cdot \alpha(n)\}. \qquad (3.8)$$

The value function is the maximum of a finite set of linear functions, and is piecewise-linear and convex for all n. ■

All exact POMDP algorithms rely on this result. Instead of having to evaluate all possible points in the state space, solving a POMDP concerns finding the set of α-vectors for each stage necessary to compute the value function.

2. Computational Complexity

We can often gain insight into the difficulty of solving a problem by analyzing its *worst-case computational complexity*, that is, the worst-case performance of the best algorithms available to solve the problem. A large literature is available on this type of classification, with the emphasis being on determining the computational difficulty of a problem.

Papadimitriou and Tsitsiklis (1987) have written the seminal paper on the computational complexity of MDPs and POMDPs. In this paper, they show that the finite-horizon POMDP is a member of a class of problems called "PSPACE-complete." This term may be less familiar than the more common classification of "NP complete," but it turns out to be a stronger indicator that a problem is intractable.

We do not define PSPACE-completeness formally here, as that would require an entire chapter. However, we want to give the reader a feel for the consequences of having the POMDP branded as PSPACE-complete. To this end, we briefly discuss the more common classes P, NP, and NP-complete, and then compare them to the class PSPACE-complete.

The class P is defined as the class of problems that can be solved in *polynomial time*, that is, computing time that is bounded by a polynomial function of the "size" of the input by a deterministic algorithm for any instance of the problem. A deterministic algorithm is what we normally define as an algorithm: a finite sequence of unambiguous instructions, each requiring finite time and storage to execute.

To describe the class NP, we must introduce the notions of a *decision problem* and a *nondeterministic algorithm*. In the most abstract sense, a decision problem is set of instances which contains a subset of so-called "yes" instances. Solving a decision problem means asking, for a generic instance, whether or not it belongs to the subset of "yes" instances. For example, we may have constructed an integer program that we want to analyze. To view this as a decision problem, we must ask questions in a form such as "does there exist a feasible solution to this integer program with an objective function value < 2.387?"

Suppose we have a mechanism that can guess at answers to a decision problem. Furthermore, suppose we also have a deterministic algorithm that can verify whether the guess answers the decision problem. The combination of the guess mechanism and the verification procedure is called a nondeterministic algorithm, and we classify the computing time of such an algorithm by considering only the time required by the verification. The class NP is the set of all problems that can be solved by polynomial-time nondeterministic algorithms.

Precisely defining a nondeterministic algorithm requires a great deal of careful development, and is not necessary here. The concept is just a mechanism used to distinguish between problems solvable in polynomial time (the set P) and problems whose solutions can be verified in polynomial time (the set NP). A commonly-used example of the difference is the traveling salesman problem, where a salesman, starting from his home, has a set of cities to visit and has fixed travel distances between each pair of cities. The objective is to find the minimum-length route that starts and returns the salesman home while visiting each city only once. No known polynomial-time algorithm exists that can solve this problem. However, suppose we recast the question as "does there exist a route that is less than 500 miles long?" While this matches the definition of the decision problem above, no available algorithm exists that can answer this question in polynomial time either. Nevertheless, a polynomial-time nondeterministic algorithm could guess a route and then *verify* whether it is less than 500 miles long in polynomial time; as a result, the traveling salesman problem is a member of NP.

Within the class NP, there are set of problems called "NP-complete," which are the hardest problems in the class NP. The common conjecture is that problems in this class are intractable; that is, they are so hard that no deterministic algorithm exists that can solve them in polynomial time. We can see from the definitions that P is a subset of NP. However, no one has been able to prove $P \neq NP$, so we cannot be sure the set $NP - P$ containing NP-complete problems is not empty. Consequently, any classification of a problem as NP-complete relies on assuming that $P \neq NP$.

Without raising any more issues (for example, defining exactly what is meant by the "size" of the input *m*), we note that NP-completeness is concerned with the *time* required to solve a problem. A similar definition exists that addresses the amount of *storage space* required to solve a problem. If a problem can be solved by a deterministic algorithm that requires storage bounded by a polynomial function of its input (polynomial space), it belongs to a class known as PSPACE. Furthermore, the classes P and NP are subsets of PSPACE.

The classification of PSPACE-complete is analogous to the classification of NP-complete; it is the class of the hardest problems in PSPACE. Garey and Johnson (1979, p. 172) note that a PSPACE-complete problem is not likely to be in P, nor is it likely to be in NP. Since NP is a subset of PSPACE, PSPACE probably contains problems harder than those in NP.

If the latter is true for a particular problem, it means we cannot even verify a guess at an answer in polynomial time, which is a rather sobering result. This would appear to condemn the overall approach of this dissertation, as we are not just solving a single PSPACE-complete problem; we are solving multiple sequences of them.

We do not cite the complexity of POMDPs as a reason to simply give up, since the sensor-shooter problem, and many other problems of this class, will not vanish just because they are provably difficult. Instead, we follow the advice of Garey and Johnson (1979, pp. 3-4) in our eventual choice of a POMDP algorithm:

> ... the knowledge [that a problem is PSPACE-complete] does provide valuable information about what lines of approach have the potential of being the most productive. Certainly the search for an efficient, exact algorithm should be accorded low priority. It is now more appropriate to concentrate on other, less ambitious approaches ... You might look for algorithms that, though not guaranteed to run quickly, seem likely to do so most of the time. Or you might even relax the problem somewhat, looking for a fast algorithm that merely finds designs that meet *most* of the component specifications.

3. Structural Characteristics

We conclude this section by noting that the POMDP, in general, does not offer much in the way of "structure" that can be exploited. In solving MDPs, it is often useful to investigate the characteristics of the optimal solutions, as they can sometimes lead to much simpler solution procedures. An example of this is work done by Iglehart (1968) on determining periodic-review inventory policies, in which he was able to show the optimal policy was always a member of a small class.

In some cases, a POMDP model yields a structural result. Ross (1971) formulated a machine maintenance/replacement problem as a POMDP for a machine with 2 states (functioning or not) and 3 possible actions: do nothing (a_1), replace (a_2), or inspect (a_3) with perfect information resulting on the machine's state (a_3). He was able to show that the optimal policy has at most 4 regions, and the "inspect" and "replace" regions are convex (Figure 3.1). With this structure, we could use a simple line search to find the optimal policy.

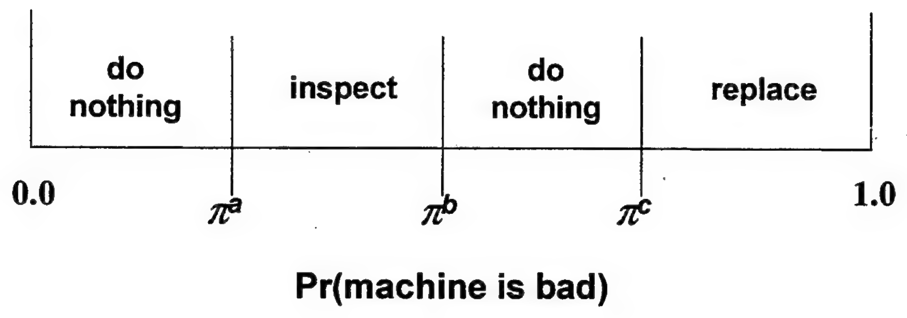

Figure 3.1: Optimal control policy for Ross's machine replacement problem. Ross (1971) showed that this machine-replacement POMDP had the structural solution shown above for a machine with two states (good or bad). A simple line search for the points π^a, π^b, and π^c finds the optimal policy.

Unfortunately, the POMDP usually destroys structure. Monahan (1980) considered an optimal stopping problem with partial observability, and showed that the solutions were not well-structured. Ehrenfeld (1976) extended Ross's model to allow for errors in inspections, and was unable to prove any structural results. Monahan (1982, p. 9) cites a long chain of papers attacking this problem, noting that White (1978) was finally able to show structural results under certain unintuitive conditions; the last word on this issue was provided by Lovejoy (1987).

The literature shows even if the optimal POMDP solutions follow an exploitable pattern (and Ehrenfeld's problem is a *very* simple POMDP), that pattern has proven difficult to obtain. In fact, we have been similarly frustrated in finding useful structure in solutions for the sensor-shooter problem modeled in Chapter IV. We do not recommend ignoring the issue of structure, as it can provide important insights and can greatly simplify solution procedures. However, documented experience indicates that we have to rely on a sound algorithm, rather than model-specific characteristics, to solve a POMDP.

B. CLASSIFICATION OF POMDP SOLUTION TECHNIQUES

The geometric implication of the piecewise-linearity of the value function is that the state space ends up being divided into convex regions, with a particular α-vector covering each region. For an object with two possible states, the belief space for a particular stage can be represented in R^1 (since $\pi_1 = 1 - \pi_2$); an example is shown in Figure 3.2. As demonstrated in the figure, the regions associated with a particular action may not be convex, and a region associated with an action may require more than one α-vector to describe the value function.

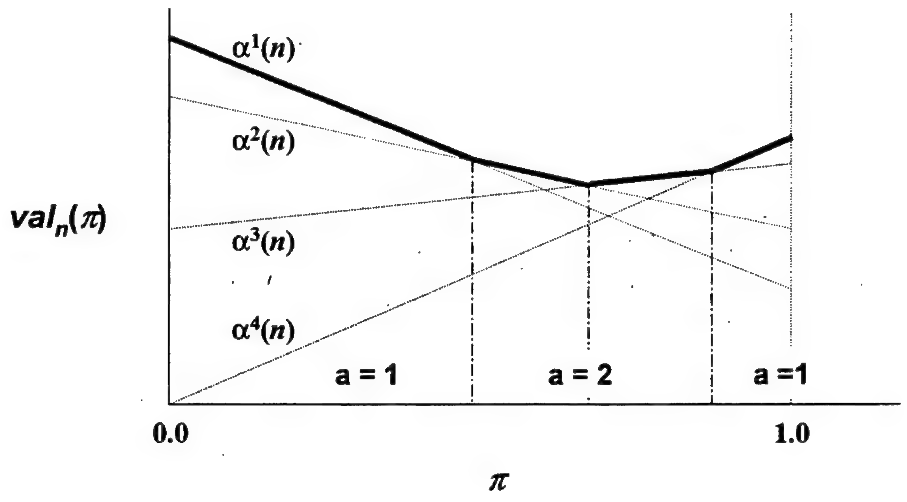

Figure 3.2: Sample Value Function for a Two-State POMDP. This value function is piecewise-linear and convex, and is stored as a set of 4 α-vectors. The policy has four regions, but the middle two regions both require the same action. Also, the region corresponding to action *a* = 1 is not convex.

Problems with higher-dimensional state spaces quickly become more difficult. Consider a case where $|E| = 3$, so that the belief space can be represented by a triangle in R^2. The components of π can be represented by the perpendicular distance from each side of the triangle, so the middle point is {1/3, 1/3, 1/3}. Each α-vector covers a partition of the triangle, and the optimal policy may require many of these partitions. Figure 3.3 illustrates an example policy and the potential difficulty of finding all these partitions.

This section is intended only to give a brief introduction to the solution algorithms. The details of many of the exact algorithms must be studied carefully, as many require solving large collections of LPs or use other search methods to find extreme points of convex polyhedra. For the exact methods, Cassandra (1994) is perhaps the most comprehensive (and certainly the most readable) source, although it must be supplemented with Cassandra, Littman, and Zhang (1997) for development of the incremental pruning methods. For the approximate methods, good sources are Lovejoy (1991) and Hauskrecht (1998).

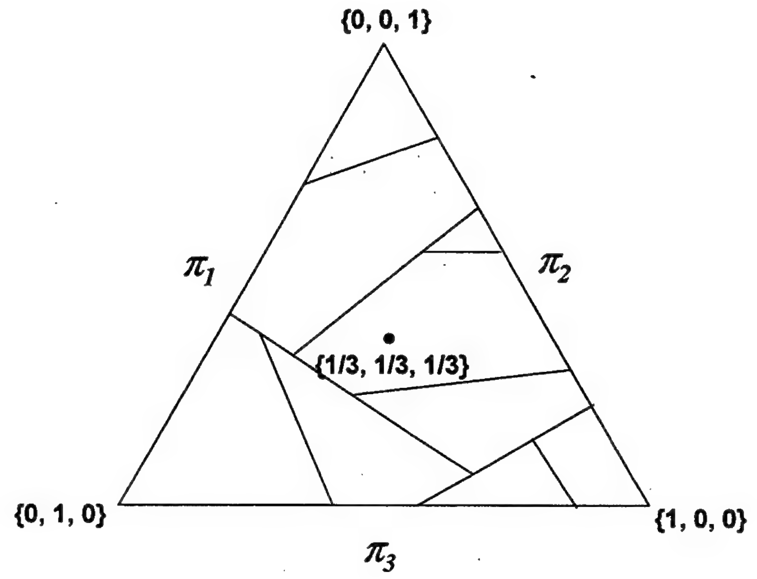

Figure 3.3: Sample State Space Partitions for a Three-State POMDP. This graphic shows the potential difficulties in determining the optimal policies for POMDPs with higher-dimensioned state spaces. The perpendicular distances from each side to each point in the triangle give the state probabilities $\{\pi_1, \pi_2, \pi_3\}$, and each region is associated with a different α-vector. A POMDP solution algorithm must find all these regions and their associated α-vectors.

1. Exact Enumerative Approaches

All POMDP solution methods rely on building the optimal n-stage policy from the optimal n-1st –stage policy. The optimal policy for stage $n = 0$ is given by definition, so the question is how to build the policy for succeeding stages.

There are several intuitive approaches for how to find the value function and the optimal policy. The first approach exploits the proof of Theorem 3.1, which demonstrates that there are only a finite number of possible α-vectors. Suppose $V(n\text{-}1)$ contains the set of α-vectors necessary to compute the optimal value function for stage n-1. Using (3.7), we can generate the set $VM(n)$, and then compute the value function using (3.8).

However, this approach can generate many vectors. If any action-observation combination is possible, then $VM(n)$ contains $|A||V(n-1)|^{|\Theta|}$ vectors for the nth stage, many of which are dominated by other α-vectors. While this may only be an inconvenience in stage n, retaining the extra vectors makes doing a similar enumeration in the next stage much more difficult. If $VM(n)$ is not reduced, then the number of vectors for each stage is given by

$$\begin{aligned}
&|VM(1)| = |A|, \\
&|VM(2)| = |A||VM(1)|^{|\Theta|} = |A|^{|\Theta+1|}, \\
&\vdots \\
&|VM(n)| = |A|^{\sum_{m=1}^{n} |\Theta|^{m-1}}.
\end{aligned} \tag{3.9}$$

We would have to generate 9.22 x 10^{18} vectors for the 7th stage of a 2-action, 2-observation POMDP using this scheme, so it is clearly unworkable.

There have been two proposals for reducing the set of α-vectors. The first was developed by Monahan (1977) and involves solving an LP for each α-vector to see if it provides a maximal value for any $\pi \in \Pi$. A simpler reduction scheme was proposed by Eagle (1984), who compares each component of each new vector to previously-generated vectors. If an existing vector's components are all strictly greater than the new vector's, the new vector can be discarded. While this approach cannot detect vectors that are dominated by combinations of other vectors, it is very simple.

The following is a general algorithm for this *exact enumerative* approach for a T-stage POMDP:

1. **Let $V(0) = \{ (r_e) \}$. Let $n = 1$.**
2. **Generate $VM(n)$ using (3.7), indexing the vectors from 1 to $|VM(n)|$.**
3. **Let $k = 1$.**
4. **Using linear programming (Monahan), dominance testing (Eagle), or both, determine if there exists some value of $\pi \in \Pi$ for which $\alpha^k(n)$ is the maximal vector. If none exists, remove the vector from $VM(n)$.**
5. **Let $k = k+1$. If $k \leq |VM(n)|$, go to 4.**
6. **Let $n = n + 1$ and let $V(n) = VM(n)$. If $n \leq T$, go to 2; otherwise, exit.**

The exact enumerative approach offers simplicity and gives good results when the numbers of actions, observations, and the time periods are small. However, the enumeration step can result in huge $VM(n)$ sets for problems with long time horizons or many possible observations.

2. Exact Constructive Approaches

Another approach is based on building up the optimal value function from a subset of the possible α-vectors. In this approach, the solution algorithm finds the optimal α-vector for a particular point in the belief space Π and gradually builds up the optimal policy by exploring more points. These *exact constructive* algorithms include those developed by Sondik (1971), Cheng (1988), Littman (1994), and Cassandra, Littman, and Zhang (1997).

Sondik's one-pass algorithm. This was the first exact POMDP method, and is summarized as follows for a *T*-stage problem:

1. **Determine the value function for $n = 0$. Let $n = 1$.**
2. **If $n > T$, exit. Otherwise, initialize the search list with a single point π in the state space. Find the optimal α-vector for this point, initialize $V(n)$ to contain this vector, and go to 4.**
3. **Remove a point from the search list, and find the optimal α-vector. If this α-vector is not in the set $V(n)$, add it to $V(n)$ and go to 5.**
4. **If no points remain, the policy for this stage is optimal. Add 1 to n and go to 2.**
5. **Use linear programming to find a region in Π for which this α-vector is guaranteed to be optimal.**
6. **Select points on the boundary of this region and add them to the search list. Go to 3.**

The use of linear programming in step 5 is common among the exact algorithms. The objective of the optimization is to find a point π^* that is an extreme point of the region covered by the current α-vector, so the focus is on formulating constraints to find such a point. Most of the refinement is concerned with efficiently generating the appropriate constraints.

The one-pass algorithm has deficiencies. As pointed out by Mukerjee and Seth (1991), the algorithm as proposed by Sondik can fail to detect regions due to the LP formulation used to determine points to add to the search list. This destroys the optimality of the algorithm and is a fatal flaw. Mukerjee and Seth present a modification to correct this problem, but Sondik's method still suffers because it tends to pick regions too conservatively. As a result, the algorithm explores many points that are not actually on region boundaries and requires much computational time on typical problems. Interestingly enough, the single real-life POMDP application we cited in Chapter 2 used Sondik's uncorrected one-pass algorithm to compute policies (Lane 1989, p. 247).

Cheng's relaxed region algorithm. Nonetheless, Sondik's algorithm was the first finite exact method proposed, and subsequent algorithms are based on the insights Sondik provided.

Cheng (1988) offered the first substantial improvement in an unpublished dissertation. Lovejoy (1991, pp. 53-54) provides the only published description of Cheng's work; however, Cassandra (1994, pp. 72-80) covers Cheng's contributions in detail in an unpublished technical report.

Cheng offered two new algorithms. The first, called the "relaxed region" method, improved on Sondik's algorithm by exploring regions larger than those proposed by Sondik around each point in the search list. Cheng was able to offer empirical evidence that this method works much faster than Sondik's, and the algorithm itself is very similar to Sondik's except for the way it searches regions. The relaxed region method does not use linear programming to find new points to search; instead it employs a method developed by Mattheis (1973) to find all vertices of a convex polytope. This avoids the potential problems in the original Sondik algorithm and offers better computational performance.

Cheng's linear support algorithm. An unfortunate characteristic of both Sondik's one-pass and Cheng's relaxed region methods is that they must run to completion for each stage. Since they build up the optimal policy region by region, they cannot be stopped prior to searching all the points in the search list. Otherwise, there is no policy or value function specified for the unexplored regions. To allow for approximate solutions, Cheng also developed another algorithm, called the "linear support" algorithm. This algorithm relies on the observation that the maximum difference between the optimal (convex) value function and a piecewise-linear lower bound occurs at the extreme points of the regions defined by the approximation (i.e., the corners of the regions in Figure 3.3; also see Theorem 3.2). Therefore, this algorithm systematically approximates the entire value function by generating an optimal vector (or linear support, hence the name), at each extreme point, stopping whenever either a tolerance is met or all extreme points have been searched.

As a result, the linear support algorithm can be stopped any time during its search, because, unlike the one-pass and relaxed region algorithms, it maintains an admissible policy over the entire state space at every step. Furthermore, we show in Section III.D that we can compute an upper bound on the difference between the solution we get when we stop the algorithm prematurely and the optimal value function. We exploit this characteristic in Chapter IV.

The witness algorithm. Littman (1994) introduced a new solution algorithm. This procedure, called the "witness" algorithm, takes a different approach to finding regions for vectors in the state space. Suppose that we have computed the optimal value function for stage *n*-1, but we now restrict the set of actions *A* to a single action *a*. We can still compute a (probably suboptimal) value function for stage *n* by only using this action. Let *V*(*n*-1) be the set of vectors defining the optimal policy for stage *n*-1, and assume we have constructed this set by considering all actions in all *n*-1 stages. In stage *n*, we reduce the set of available actions *A* to a single action *a*, and solve (3.3) for the resulting value function, denoted $val_n^a(\pi)$. Let $V^a(n)$ be the minimal set of vectors necessary to compute $val_n^a(\pi)$, so $val_n^a(\pi) = \max_{\alpha(n)\in V^a(n)} \{\alpha(n)\cdot\pi\}$. If we construct these value functions for every $a \in A$, we can write the optimal value function for stage *n* as

$$val_n(\pi) = \max_{a\in A}\{val_n^a(\pi)\}. \qquad (3.10)$$

This representation exploits the fact that fixing the action, determining the resulting value functions $val_n^a(\pi)$, and combining them can be done more quickly than optimizing over all actions at each point we choose to test in the belief space. It also takes advantage of the fact that a particular action may be optimal in a single convex region in state space, but the value function may be described by several different α-vectors over the same region (see Figure 3.2).

The witness algorithm takes its name from the way it computes the value function for each action. The procedure for determining $val_n^a(\pi)$ and its minimal set of vectors $V^a(n)$ is similar to Sondik's one-pass algorithm, but since the action is fixed, the linear program that looks for points where the current approximation differs from the optimal value function (the so-called "witness" points) is much more efficient. The algorithm terminates with a reduction step identical to that used in the exact enumerative algorithms to reduce $V(n) = \bigcup_{a \in A} V^a(n)$ to the minimal set of α-vectors necessary to define *val$_n$*(*π*). However, $\bigcup_{a \in A} V^a(n)$ generally contains many fewer vectors than enumerating all possible α-vectors, so the witness algorithm is a substantial improvement.

The incremental pruning algorithm. The latest development in POMDP solution algorithms is a method known as "incremental pruning" (Cassandra, Littman, and Zhang 1997). Incremental pruning continues the philosophy of the witness algorithm by defining a set of value functions for each action and observation, combining them into a value function for each action, and then combining those functions to compute the optimal value function.

Incremental pruning is actually a family of algorithms, and is based on the following decomposition of the DP recursion (3.3):

$$val_n(\pi) = \max_a \{val_n^a(\pi)\}, \tag{3.11}$$

$$val_n^a(\pi) \equiv \sum_{\theta \in \Theta} val_n^{a,\theta}(\pi), \tag{3.12}$$

$$val_n^{a,\theta}(\pi) \equiv \frac{-R_a}{|\Theta|} + \Pr(\theta | \pi, a) val_{n-1}(Tr[\pi | a, \theta]). \tag{3.13}$$

This decomposition extends the witness algorithm by building value functions for each action and observation outcome, combines them into a value function for each action, and finally combines those value functions into the overall value function. Since each of the expressions

(3.11), (3.12), and (3.13) can be expressed as maxima of a finite set of linear functions, the algorithm builds the linear functions "incrementally."

The notion of pruning comes from the mechanisms that are used to create the minimal sets of vectors describing each of these value functions. We do not cover the details here, other than to note that the authors describe a "purge" operator that can be used to efficiently remove dominated vectors from the various sets as the value function is constructed. Linear programs are necessary to find witness points for computing the minimal vector sets for (3.11), (3.12), and (3.13), but since dominated vectors are discarded as each set is constructed, the algorithm is potentially more efficient than the witness algorithm.

Incremental pruning is a family of methods because there are some choices in how to implement the incremental pruning operation. At one extreme, the algorithm is identical to Monahan's enumeration method. However, the authors investigate other implementations that substantially improve the algorithm's performance on a set of common POMDP test problems, and show that incremental pruning with appropriate tuning gives the best results on their test problems (see Table 3.2).

Figure 3.4 shows the genealogy of the exact algorithms. We note that Monahan (1977) credits his enumerative algorithm to Sondik, but his enumerative procedure is completely different from the one-pass algorithm. Incremental pruning borrows ideas from both the enumerative and constructive methods, and has ancestors in both branches of the tree.

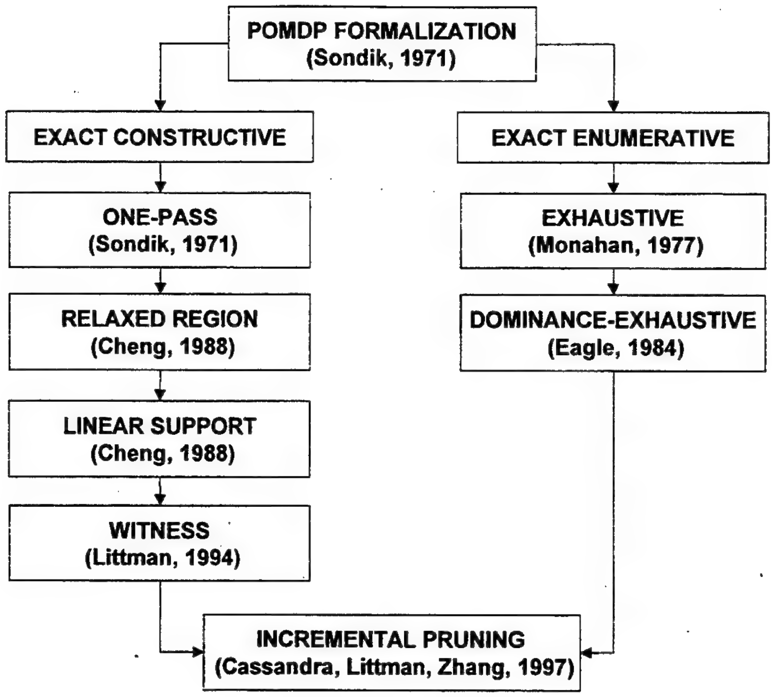
POMDP FORMALIZATION (Sondik, 1971)
EXACT CONSTRUCTIVE
EXACT ENUMERATIVE
ONE-PASS (Sondik, 1971)
EXHAUSTIVE (Monahan, 1977)
RELAXED REGION (Cheng, 1988)
DOMINANCE-EXHAUSTIVE (Eagle, 1984)
LINEAR SUPPORT (Cheng, 1988)
WITNESS (Littman, 1994)
INCREMENTAL PRUNING (Cassandra, Littman, Zhang, 1997)

Figure 3.4: Genealogy of Exact POMDP Algorithms. This chart shows the historical development of the current exact finite-POMDP algorithms, and also depicts the conceptual relationships among them.

3. Approximate Approaches

The approximate algorithms do not attempt to find an exact solution, but instead concentrate on finding near-optimal policies with bounds on the value function. One of these, the linear support algorithm developed by Cheng (1988), is also an exact constructive method. However, the others, including those proposed by Eckles (1968), Lovejoy (1991), and Hauskrecht (1998), convert the POMDP to an MDP by approximating the continuous state space with a set of grid points. These grid-based algorithms work as follows:

1. **Determine $val_0(\pi)$ and choose a set of grid points $G = \{g: g \in \Pi\}$. Let $n = 1$.**
2. **If $n > T$, go to 4. Otherwise, for each $g \in G$, find the α-vectors for the lower-bound value functions, and upper-bound values.**
3. **Add 1 to n and go to 2.**
4. **Find the error bounds by interpolating the upper-bound values for all starting states of interest.**

Step 2 is what distinguishes various grid approaches. For the lower bound, note that if we only find the optimal α-vectors for each $g \in G$, then we can write the value function for a particular stage as

$$\underline{val}_n(\pi) = \max_g\{\alpha^g(n)\cdot\pi\}. \tag{3.14}$$

This is a legitimate value function for all π, and can be substituted into the DP recursion (3.3) to compute a policy for the next stage as follows

$$\underline{val}_n(\pi) = \max_{a\in A}\left\{-\pi\cdot R_a + E\left[\max_g \alpha^g(n-1)\cdot Tr(\pi|a,\theta)\right]\right\}, \quad n = 1,2,\ldots T. \tag{3.15}$$

Lovejoy (1991, pp. 165-166), shows rigorously that the DP recursion preserves this bound.

The lower bound requires a fixed amount of computation since the number of grid points is fixed, and it is necessary to store the vectors for each grid point and stage. However, computing the upper bound (which is an important issue in this dissertation) requires only storing the upper bound values at each iteration point. This, along with some specified interpolation function, is used to determine the value function for points $\pi \notin G$. Therefore, the computation of the upper bound would be

$$\overline{val}_n(g) = \max_a\left\{-\pi\cdot R_a + E\left[Inter\left\{\overline{val}_{n-1}\left(Tr\left[g|a,\theta\right]\right)\right\}\right]\right\}, \quad n = 1,2,\ldots T. \tag{3.16}$$

The *Inter* function must be provided to determine the upper bound value for the transformed point g, since this point is probably not a member of G. Lovejoy (1991, pp. 166-168) also shows that for his particular interpolation scheme and grid selection, the DP recursion preserves upper bounds.

Figure 3.5 is a revision of Figure 3.2, showing the optimal policy, a lower bound for a fixed grid, and an upper bound using a simple interpolation scheme for stage $n = 1$. Current research (Hauskrecht 1998) focuses on fast interpolation schemes, bound improvement, and variable grid spacing, while Lovejoy (1991) derives his results from using grids with regular spacing.

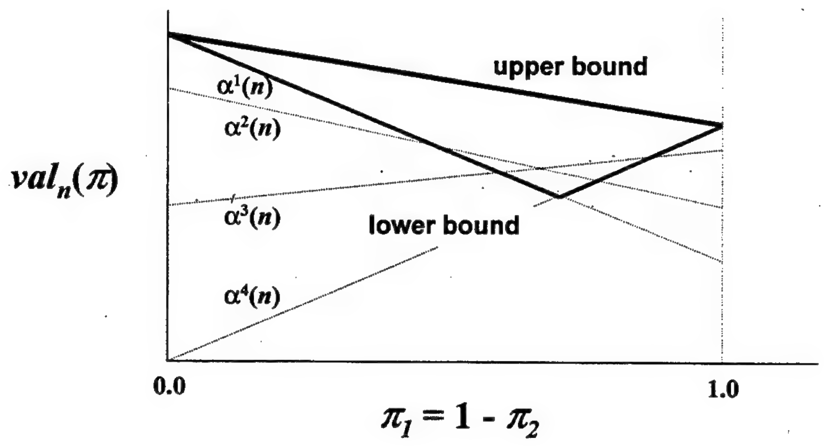

Figure 3.5: Example Upper and Lower Bounds for a Grid Algorithm in Stage n = 1. This shows the same function as Figure 3.2, except that the value functions are computed using 2 grid points, at 0.0 and 1.0. The lower bound is an admissible policy and consists of the α-vectors computed at the grid points. The upper bound is computed via interpolation, and the error depends on the state.

Grid methods have considerable attraction for POMDPs. In addition to generating legitimate policies, they provide bounds on the value function. Also, they allow the amount of work necessary to complete the algorithm to be specified in advance, as the choice of the grid completely determines the number of function evaluations and DP updates required. This is in stark contrast to the exact algorithms, which must uncover all the regions in the state space and cannot predict in advance the number of points they search in each stage. For many large POMDPs, and particularly POMDPs with long time horizons, grid methods are probably the only alternative.

C. CHOOSING A POMDP ALGORITHM FOR THE DECOMPOSITION

With these (short) descriptions of the POMDP solution methods available, the question is which should be used in the decomposition algorithm. In the POMDP literature, the assumption is a fixed cost structure and the emphasis is on solving a particular POMDP. In the decomposition, we must repeatedly solve a potentially large number of POMDPs (one for each of the $|J|$ classes), and the changes in the resource costs λ virtually ensure that the previous POMDP solution does not apply.

We first consider the exact algorithms. Cheng (1988, p. 64) reports the solution times shown in Table 3.1 for a machine-replacement problem with $|A| = 4$ actions, $|\Theta| = 2$ observations, $T = 20$ time periods, and $|E| = 3$ states for the one-pass, enumerative, relaxed region, and linear support algorithms. The empirical performance of the linear support algorithm is superior for the problems Cheng considers.

Algorithm	Enumerative	One-Pass	Relaxed Region	Linear Support
Run Time	0.937	2.947	0.894	0.751

Table 3.1: Runtimes for Sondik's Machine Replacement Problem (Cheng 1988, p. 64). This problem has $|A| = 4$ actions, $|\Theta| = 2$ observations, $T = 20$ time periods, and $|E| = 3$ states, and the times are CPU seconds on an Amdahl 5860 computer. The linear support algorithm is the fastest in this case, and in the other POMDPs in Cheng's dissertation.

However, Cassandra, Littman, and Zhang (1997, pp. 59-60) report total running times for the enumerative, linear support, witness, and incremental pruning algorithms for a much more difficult set of test problems. The problem sizes and run times in CPU-seconds on a SPARC-10 workstation are shown in Table 3.2, with the exception of the linear support algorithm. The authors state that the memory requirements for the linear support algorithm exceeded the capacity of their workstation for all these problems.

Problem	States	Actions	Observations	Stages	Run Times (CPU Seconds)		
					Witness	IP (RR)	Enumerative
1D maze	4	2	2	70	9.3	2.3	2.2
4x3	11	4	6	8	727.1	157.0	DNF
4x4	11	4	11	367	3226.0	909.2	216.7
Cheese	16	4	2	374	351.8	203.3	DNF
Painting	11	4	7	373	5608.4	5226.4	1116.9
Network	4	4	2	371	6422.9	722.5	DNF
ID	7	4	2	14	417.0	166.0	DNF
Shuttle	8	3	5	4	1676.7	145.9	DNF
4x3 CO	12	6	5	4	24.6	22.7	DNF

Table 3.2: Runtimes for Various POMDPs Reported by Cassandra, Littman, and Zhang (1997, pp. 59-60). In these test problems, the "RR" variant of incremental pruning provided the best performance. DNF (Did Not Finish) indicates that the enumerative algorithm was unable to solve the problem in less than 8 CPU hours. The linear support algorithm was unable to solve any of these problems due to memory limits on the researchers' SPARC-10 workstation.

Table 3.1 and Table 3.2 yield several insights into the nature of POMDP models. Clearly, as the number of states and stages grow, the difficulty of obtaining an exact solution increases very quickly. Indeed, if a problem as difficult as the "Painting" problem in Table 3.2 were embedded in the decomposition developed in this dissertation, the total running times would be intolerable.

Yet, Table 3.2 presents a very pessimistic view of what is possible. The authors of that paper did not document any attempt to run the linear support algorithm as an approximate method, but instead set it to run at the same numerical tolerances as the other algorithms. Consider what Cheng (1988, p. 66) reports when he adjusts the "tolerance factor" (to be explained in detail in Section III.D.2) for the linear support algorithm, as shown in Table 3.3.

Cheng shows the linear support algorithm finds solutions within 1.7% of the optimum value with a 76% decrease in runtime, and optimal solutions with a 7.5% decrease in run time, when run in an approximate mode.

Linear Support Tolerance Setting	Run Time CPU-sec	Percent Runtime Decrease	Maximum Absolute Error	Maximum Percent Error
0.1	0.179	76.2%	0.1251	1.74%
0.01	0.536	28.6%	0.0086	0.12%
0.005	0.604	19.6%	0.0028	0.04%
0.001	0.695	7.5%	0.0000	0.00%

Table 3.3: Runtime Reductions and Maximum Errors for Sondik's Machine Replacement Problem (Cheng 1988, p. 66) Using the Approximate Linear Support Algorithm. This table shows the maximum errors in the value function for the same machine-replacement problem cited in Table 3.1. The tolerance setting is the maximum difference in the optimal value function and the approximate value function per stage. By adjusting this setting, large runtime reductions are possible with very little loss of accuracy.

Lovejoy (1991, p. 173) solves the same machine-replacement problem using a grid scheme, and, for a problem with $T = 11$ stages, achieves a maximal error in the value function of 1.3% while only using 66 grid points. Furthermore, Lovejoy is able to solve a $T = 100$ version of the same problem using 231 grid points with a maximum error of only 0.2%. While Lovejoy does not report any solution times, they are probably minimal for these grid sizes.

Having viewed some empirical results, reconsider what is needed in the decomposition. To determine the upper bound, we need to find the value of the optimal policy over all of S for each of the $|J|$ object classes. Rewriting (2.20) and (2.21) shows this relationship:

$$u(S;\lambda) = \sum_{i \in I} \lambda_i b_i + \sum_{j \in J} N_j \, val_T^j\left[\pi^j(1), \lambda\right]. \tag{3.17}$$

From (2.6), a particular policy is potentially improving if its reduced cost is positive; that is,

$$val_T^j\left[\pi^j(1), \lambda\right] - w_j > 0 \Rightarrow \text{ possible improvement}. \tag{3.18}$$

However, a POMDP algorithm that provides an upper bound still suffices for the decomposition upper bound computation:

$$\begin{aligned} u(S;\lambda) &= \sum_{i \in I} \lambda_i b_i + \sum_{j \in J} N_j \, val_T^j\left[\pi^j(1), \lambda\right] \\ &\leq \sum_{i \in I} \lambda_i b_i + \sum_{j \in J} N_j \, \overline{val}_T^j\left[\pi^j(1), \lambda\right]. \end{aligned} \tag{3.19}$$

Also, an algorithm that computes a lower bound corresponding to an admissible policy can still improve the master LP:

$$\underline{val}_T^j\left[\pi^j(1), \lambda\right] - w_j > 0 \Rightarrow \text{possible improvement}. \tag{3.20}$$

Also, we are not solving to exact optimality, as the ε-optimal decomposition allows some gap between the upper and lower bounds. **This is the key point in our choice of a POMDP algorithm for the decomposition algorithm: an exact method is not required.** So long as we can generate tight upper bounds on the value function and provide admissible improving policies, there is no need to invest in the computational overhead required by the exact methods.

Furthermore, the resource prices change rapidly as the decomposition algorithm progresses. We show examples of this behavior in Chapter IV, but the point here is that solving a POMDP to optimality in the early part of the decomposition is simply unreasonable, because the master LP only needs *improving* columns. Most commercial LP packages, such as OSL (IBM 1992, p. 81) take advantage of this by using much faster column pricing schemes in the early stages of the simplex algorithm, because objective function improvements come easily in early iterations. Only later do these packages switch to more effective, but more computationally-intensive, column pricing mechanisms.

An approximate POMDP algorithm offers the same advantages. By controlling some set of error parameters (such as the grid size or Cheng's tolerance factor), we can control the tightness of the upper bound on the value function and reduce it as necessary. However, the aim in the decomposition is to minimize the time required to find an improving policy (column).

With this in mind, we can choose from two types of approximate methods: the linear support algorithm or the grid algorithms. In the linear support algorithm, we can designate a maximum error gap (difference between the upper and lower bound), but we cannot predict the computational work required in advance. In the grid methods, we can specify the computational work required by our choice of a grid size, but we cannot predict in advance the maximum error gap.

Since the decomposition terminates using an error gap tolerance, we have chosen the linear support algorithm. It works well for the sensor-shooter problem (which only has two states) and can be governed by any desired tolerance. However, this method may not be a good choice for POMDP subproblems with large numbers of states, observations, or time periods. In those cases, using a grid is probably a better alternative. Nonetheless, we are firm in recommending that approximate methods are better for the decomposition algorithm, and this position is reinforced by our empirical results in Chapter IV.

D. THE LINEAR SUPPORT ALGORITHM

Early in our research, we noticed a two-paragraph discussion in Sondik (1971, pp. 51-52) on a possible alternative algorithm for solving POMDPs. Sondik did not explore this method, nor had we seen any mention of it in the published OR literature. We began developing the algorithm for the sensor-shooter problem, and developed a convergence proof as well as bounds. We speculated that we had found a new solution algorithm for POMDPs.

However, a chance conversation at a conference alerted us to work of Cheng (1988), who had already developed the method we thought we had discovered. We are a decade too late to claim the result, but we present the algorithm in detail to show its usefulness in the overall decomposition.

1. The Basic Algorithm: Description and Convergence

Let $U^*(n)$ be the minimal set of α-vectors needed to describe the value function for stage n. As before, the POMDP value function can be represented as

$$val_n(\pi) = \max_{\alpha(n) \in U^*(n)} \{\alpha(n) \cdot \pi\}. \tag{3.21}$$

Suppose we have the optimal value function for the previous stage, so we can find the optimal $\alpha(n) \in VM(n)$ for any point in the belief space using (3.7). Now suppose we choose an arbitrary set of points in Π and find their optimal vectors. Let $U'(n)$ be this set of vectors, and let $val'_n(\pi) = \max_{\alpha(n) \in U'(n)} \{\alpha(n) \cdot \pi\}$ be the approximate value function defined by $U'(n)$. For any choice of $U'(n)$ and π, $val'_n(\pi) \le val_n(\pi)$, and $val'_n(\pi)$ is also piecewise-linear and convex.

Define $err_n(\pi)$ as the error between the approximation and the optimal value function for any belief state, that is,

$$\begin{aligned} err_n(\pi) &\equiv val_n(\pi) - val'_n(\pi) \\ &= \left(\max_{\alpha(n) \in U^*(n)} \{\alpha(n) \cdot \pi\}\right) - \left(\max_{\alpha(n) \in U'(n)} \{\alpha(n) \cdot \pi\}\right). \end{aligned} \tag{3.22}$$

Recall that the key to most POMDP algorithms is determining which points in the state space to search for new vectors. By inspecting Figure 3.6, it appears that if we choose any $U'(n)$ that does not contain all the vectors in $U^*(n)$, the maximum error occurs at an extreme point of one of the regions of Π associated with an α-vector in $U'(n)$. These intersections are called extreme points, and are formally defined below:

Definition 3.1: An **extreme point** of a convex set in an n-dimensional Euclidean space is a point that cannot be written as a strict convex combination of two other points in the set.

For example, suppose π_1, π_2, and π_3 are members of a convex set Π, π_1 is an extreme point, and $\pi_1 = \omega\,\pi_2 + (1-\omega)\,\pi_3$ for some $\omega \in (0,1)$. Then $\pi_1 = \pi_2 = \pi_3$.

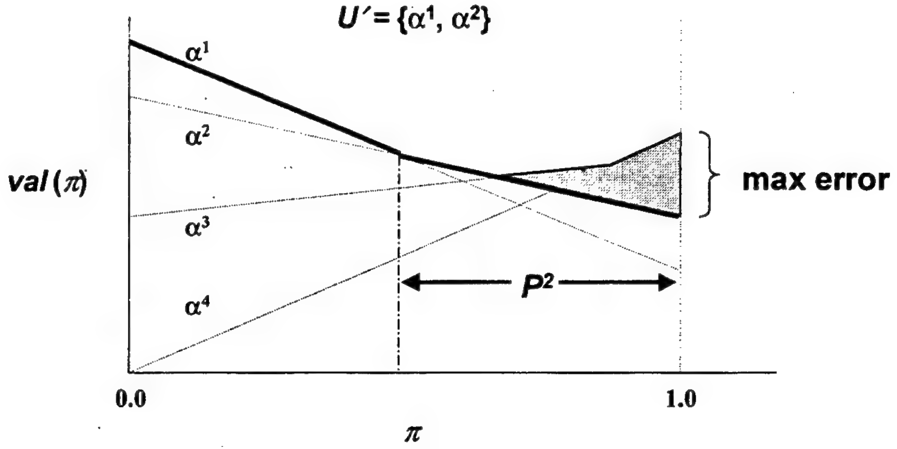

Figure 3.6: Approximate and Exact Value Functions. The approximate value function above is constructed from vectors in the set U'. The error in the approximate value function is shaded; the maximum error appears to occur at an extreme point $\pi = 1.0$ of the region P^2 associated with vector α^2.

The following theorem shows that the intuition from Figure 3.6 is correct:

Theorem 3.2 (Cheng 1988, p. 54): The maximum error occurs at an extreme point of a region $P^m(n)$ defined by $P^m(n) = \{\pi\!: \ \pi \in \Pi,\ \alpha^m(n)\cdot\pi \geq \alpha(n)\cdot\pi\ \forall\, \alpha(n) \in U'(n)\}$ for some $\alpha^m(n) \in U'(n)$.

Proof: Suppose $err_n(\pi)$ is maximized at a point $\pi_1 \in P^m(n)$. $P^m(n)$ is a polytope (a bounded polyhedral set) and is therefore a convex set. If π_1 is an extreme point of $P^m(n)$, the theorem holds. Now, suppose that π_1 is not an extreme point of $P^m(n)$. By the "representation theorem" (e.g., Bazaraa, Shetty, and Sherali 1993, p. 60), any point in $P^m(n)$ can be represented as a convex combination of the extreme points in $P^m(n)$, so $\pi_1 = \omega\pi_2 + (1-\omega)\pi_3$ for some $\omega \in (0,1)$ and $\pi_2, \pi_3 \in P^m(n)$.

For all $\pi \in P^m(n)$, $err_n(\pi) = val_n(\pi) - \alpha^m(n)\cdot\pi$. Since $val_n(\pi)$ is convex and $\alpha^m(n)\cdot\pi$ is linear, err(π) is convex over $P^m(n)$. As a result, we can write

$$err_n(\pi_1) \le \omega\, err_n(\pi_2) + (1-\omega) err_n(\pi_3). \tag{3.23}$$

However, $err_n(\pi_1) \ge err_n(\pi_2)$ and $err_n(\pi_1) \ge err_n(\pi_3)$ by assumption, so.

$$err_n(\pi_1) = err_n(\pi_2) = err_n(\pi_3). \tag{3.24}$$

Therefore, the maximum error occurs at an extreme point of $P^m(n)$. ■

Theorem 3.2 is the key to the linear support algorithm. It is only necessary to search the extreme points of the regions in Π defined by current approximation to find the maximal error; if it is 0 at all the extreme points, there is no error and the approximation is exact. Otherwise, the algorithm computes the optimal α-vector for the extreme point, adds it to the current set of vectors, computes the new extreme points, and continues.

We use (3.6) to find the maximal vector at each point π′. Suppose a' maximizes (3.6), and define the function $l[\pi, a, \theta, V(n)]$ as

$$l[\pi,a,\theta,V(n)] = \arg\max_k \left\{ \begin{array}{l} \pi\cdot\left[-\frac{R_a}{|\Theta|} + P^a\cdot\Theta^a_\theta\cdot\alpha^k(n)\right] \\ \alpha^k(n)\in V(n) \end{array} \right\}. \tag{3.25}$$

Then, the new vector $\alpha'(n)$ is given by

$$\alpha'(n) = \sum_{\theta\in\Theta} -\frac{R_{a'}}{|\Theta|} + P^{a'}\cdot\Theta^{a'}_\theta\cdot\alpha^{l[\pi',a',\theta,V(n-1)]}(n-1). \tag{3.26}$$

And, we compute the error function as

$$err_n(\pi') = \alpha'(n) \cdot \pi' - \left(\max_{\alpha(n) \in U'(n)} \{\alpha(n) \cdot \pi'\} \right). \tag{3.27}$$

By construction, $val_n(\pi') = \alpha'(n) \cdot \pi'$. Also, $val_n(\pi) \geq \alpha'(n) \cdot \pi \;\; \forall \pi$; that is, $\alpha'(n)$ is a subgradient for the optimal value function (e.g., Lovejoy 1991, p. 54). The name "linear support" refers to the fact that each new α-vector is a support for the optimal value function.

The entire linear support algorithm is as follows:

1. **Compute $val_0(\pi)$. Set $n = 1$.**
2. **Initialize the set of extreme points X to be the |E| extreme points of the belief space Π. Find the α-vectors for each $\pi' \in X$, and initialize $U'(n)$ to be the set of these vectors.**
3. **If $X = \varnothing$, go to 6. Otherwise, pick $\pi' \in X$ and compute $\alpha'(n)$ using (3.26) and $err_n(\pi')$ using (3.27); let $X = X - \pi'$.**
4. **If $err_n(\pi') = 0$, go to 3.**
5. **Find all extreme points of the region $P'(n)$ defined by $\alpha'(n)$ and add them to X. Find any extreme points dominated by $\alpha'(n)$ and remove them from X. Add $\alpha'(n)$ to $U'(n)$, and go to 3.**
6. **Set $U^*(n) = U'(n)$, and let $val_n(\pi) = \max_{\alpha(n) \in U^*(n)} \{\alpha(n) \cdot \pi\} \;\; \forall \pi$. Store $U^*(n)$.**
7. **Let $n = n+1$. If $n > T$, exit; otherwise, go to 2.**

The first 4 steps are straightforward. However, step 5 is not so simple, and is the reason that many researchers have problems with this method. As in Theorem 3.2, the new set of extreme points X' is the set of all extreme points of the following polytope $P'(n)$:

$$P'(n) = \{\pi: \;\; \pi \in \Pi, \; \alpha'(n) \cdot \pi \geq \alpha(n) \cdot \pi \; \forall \alpha(n) \in U'(n)\}. \tag{3.28}$$

Unfortunately, the number of extreme points of P' is an exponential function of the number of faces of $P'(n)$, and the number of faces of $P'(n)$ is an exponential function of $|U'(n)|$. Furthermore, the algorithm must find and test *all* extreme points in $P'(n)$, or it cannot guarantee optimality. Cheng uses methods developed by Mattheiss (1973) to find all such points. Cassandra

(1994, p. 77) unfortunately refers to these schemes as "interior-point" methods, which may cause confusion for readers who use his excellent survey of POMDP algorithms. These vertex-finding methods are not the same as interior-point methods for linear programming.

We do not explore vertex-finding methods, as the sensor-shooter problem only requires two states and we can find the extreme points with algebra. However, we caution readers that computing the extreme points of $P'(n)$ is a computationally-intensive step, and is the reason that the linear support algorithm could not solve the problems posed by Cassandra, Littmann, and Zhang (1997) that we referenced in Section III.C.

Step 5 does save some time by determining if any points currently in X are dominated by the new vector, that is, if $\alpha'(n)\cdot\pi \geq \max_{\alpha(n)\in U'(n)}\{\alpha(n)\cdot\pi\}$ for some $\pi \in X$. If a point is dominated, it is no longer an extreme point of the current approximation and does not need to be tested. Note also that we never remove a vector from $U'(n)$, because each vector in that set is optimal for at least one region in the belief space.

Figure 3.7 shows an example of the algorithm. The set of extreme points is initialized with $X = \{0,1\}$, which generate vectors α^1 and α^4. Adding α^1 creates a new partition in the belief space, and the algorithm adds the extreme point π^1 to X. The algorithm then tests at point π^1, and finds vector α^3. The region associated with α^3 adds has 2 extreme points π^2 and π^3, which are added to X. Testing these points uncovers the new vector α^2 at π^2. There is no improvement possible at the remaining extreme points, so the value function is optimal.

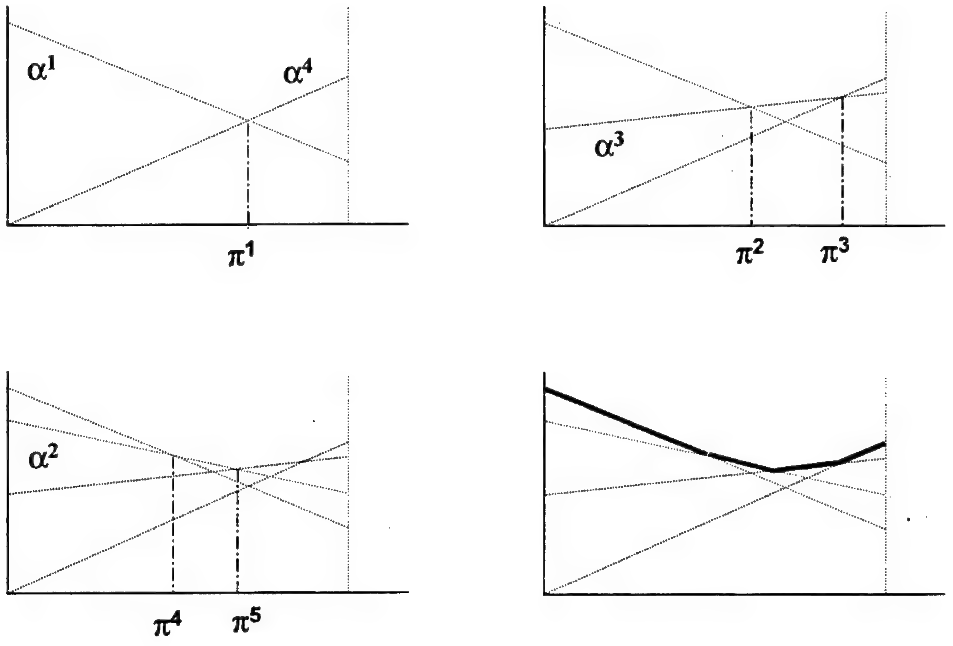

Figure 3.7: Example of the Linear Support Algorithm. The algorithm tests first at the extreme points of the state space, adding vectors α^1 and α^4. It then tests at the extreme point π^1, and finds vector α^3. This leads to two more new extreme points, π^2 and π^3; testing at π^2 uncovers the new vector α^2. Points π^4 and π^5 do not lead to further improvement, so the value function is optimal for this stage.

The following theorem establishes the convergence and finiteness of the linear support algorithm.

Theorem 3.3 (Cheng 1988, p. 61): The linear support algorithm is finite.

Proof: The proof of Theorem 3.1 shows the number of possible α-vectors for each stage is finite. Therefore, the algorithm can only construct a finite number of polytopes of the form in (3.28), and each of these polytopes has a finite number of extreme points. As a result, the algorithm can only search a finite number of extreme points in each stage. Since the number of stages is finite, the algorithm is finite. ■

2. The Linear Support Algorithm as an Approximate Method

The conversion of the linear support algorithm to an approximate method can be accomplished in two different ways. The first constrains the computations done by the algorithm by stopping when the number of α-vectors in the approximation reaches some preset limit. Cheng (1988, pp. 76-77) tested this on some sample problems, but did not investigate in any depth. There are some possibilities for such an approach (see Chapter VI), but we do not consider them in this dissertation either.

The other way to convert the algorithm to an approximate method is to change the error test in step 4. Instead of skipping the point if the error is 0, the test can be changed to skip generating an α-vector for a point if the maximum error is less than some tolerance ε:

4. If *err*(π') < ε, go to 3.

In the implementation of the algorithm, we would not normally test for an error of exactly 0, since this would be numerically unsound. However, the effect of this modification is not just to reject nearly parallel α-vectors; it can be used to produce a value function at any desired level of accuracy. The following result, which is stated without proof by Cheng (1988, p. 67), gives the overall accuracy of the approximation:

Theorem 3.4: For a *T*-period POMDP, the maximum error in the value function computed by the linear support algorithm using a tolerance ε is *T* ε.

Proof: We use induction to prove the result. Let $val_n(\pi)$ be the exact optimal value function for stage *n*, and let $V(n)$ be the minimal set of α-vectors describing $val_n(\pi)$. Let $\underline{val}_n(\pi)$ be the lower bound computed by the linear support algorithm using a tolerance of ε for stages 1,2, ... *n*, and let $\underline{V}(n)$ be the minimal set of α-vectors describing $\underline{val}_n(\pi)$.

In the recursion (3.3), $val_0(\pi)$ is given by definition and is exact. Using the linear support algorithm, we compute $\underline{val}_1(\pi)$ using tolerance ε. Since $V(0)=\underline{V}(0)$, $val_1(\pi)-\underline{val}_1(\pi)\le\varepsilon \quad \forall\,\pi\in\Pi$ by Theorem 3.2. Therefore, the result is true for $T=1$.

Define the *intermediate value function* $val_n'(\pi)$ as

$$val_n'(\pi)\equiv \max_{a\in A}\left\{-\pi\cdot R_a+E\left[\underline{val}_{n-1}\left(Tr\left[\pi\,|\,a,\theta\right]\right)\right]\right\}. \tag{3.29}$$

This means $\underline{val}_n(\pi)$ and $val_n'(\pi)$ are computed with the same set of stage *n*-1 vectors, but using different tolerances. By Theorem 3.2, the maximum difference between the two functions is ε:

$$val_n'(\pi)-\underline{val}_n(\pi)\le\varepsilon \quad \forall\,\pi\in\Pi. \tag{3.30}$$

Suppose $val_n(\pi)-\underline{val}_n(\pi)\le c \quad \forall\pi\in\Pi$. Then, by (3.3) and (3.29),

$$\begin{aligned}&val_{n+1}(\pi)-val_{n+1}'(\pi)=\\&\quad\max_{a\in A}\left\{-\pi\cdot R_a+E\left[val_n\left(Tr\left[\pi\,|\,a,\theta\right]\right)\right]\right\}-\max_{a'\in A}\left\{-\pi\cdot R_{a'}+E\left[\underline{val}_n\left(Tr\left[\pi\,|\,a',\theta\right]\right)\right]\right\}.\end{aligned}$$

Now, constrain the second maximization to pick the same action as the first. Then, we can write the following inequality:

$$\begin{aligned}&val_{n+1}(\pi)-val_{n+1}'(\pi)\\&\le\max_{a\in A}\left\{-\pi\cdot R_a+E\left[val_n\left(Tr\left[\pi\,|\,a,\theta\right]\right)\right]+\pi\cdot R_a-E\left[\underline{val}_n\left(Tr\left[\pi\,|\,a,\theta\right]\right)\right]\right\}\\&\le\max_{a\in A}\left\{E\left[val_n\left(Tr\left[\pi\,|\,a,\theta\right]\right)\right]-E\left[\underline{val}_n\left(Tr\left[\pi\,|\,a,\theta\right]\right)\right]\right\}\\&\le\max_{a\in A}\left\{E\left[val_n\left(Tr\left[\pi\,|\,a,\theta\right]\right)-\underline{val}_n\left(Tr\left[\pi\,|\,a,\theta\right]\right)\right]\right\}\\&\le\max_{a\in A}\left\{E\left[c\right]\right\}.\end{aligned}$$

Therefore,

$$val_{n+1}(\pi)-val_{n+1}'(\pi)\le c. \tag{3.31}$$

Now assume, for an arbitrary stage *n*, that $val_n(\pi) - \underline{val}_n(\pi) \le n\varepsilon$. From (3.30), $val'_{n+1}(\pi) - \underline{val}_{n+1}(\pi) \le \varepsilon$. Using (3.31), we also know that $val_{n+1}(\pi) - val'_{n+1}(\pi) \le n\varepsilon$. Adding these two inequalities means $val_{n+1}(\pi) - \underline{val}_{n+1}(\pi) \le (n+1)\varepsilon$, so the theorem is true for any *n*. Therefore, the maximum error is $T\varepsilon$. ■

This bound can be tightened by the following corollary:

Corollary 3.5: Let $\varepsilon_n \le \varepsilon$ be the largest error among any points tested and skipped by the linear support algorithm using a tolerance ε in stage *n* of a *T*-stage POMDP. Then

$$val_t(\pi) - \underline{val}_t(\pi) \le \sum_{n=1}^{t} \varepsilon_n \le t\varepsilon \quad \forall \pi \in \Pi, t = 0,1,\ldots T. \tag{3.32}$$

Proof: Substitute ε_n for ε in Theorem 3.4; the result follows. ■

For any setting of ε, the linear support algorithm computes an admissible policy. By storing the sets of vectors $\underline{V}(n)$ for all stages *n* and the actions associated with each vector in each set, we have a map from the belief space to the actions. We can use (3.5) to compute the action $a_n(\pi)$, and the value function $\underline{val}_n(\pi)$ is a lower bound on the optimal value function. We also have an upper bound on the optimal value function; that is, $val_t(\pi) \le \underline{val}_t(\pi) + \sum_{n=1}^{t} \varepsilon_n$.

Not only is $\underline{val}_n(\pi)$ a lower bound on the value function of the optimal policy, it is also a lower bound on the value function of the policy "induced" by the sets $\underline{V}(n)$. To see this, note that for each stage *n*, $\underline{V}(n)$ partitions the belief space into $|\underline{V}(n)|$ convex regions, one for each vector. The action associated with the vector is the action we take in that region, so $\underline{V}(n)$ induces a map from the belief state to the actions and defines a policy. We denote value function induced by $\underline{V}(n)$ as $valr_n(\pi)$.

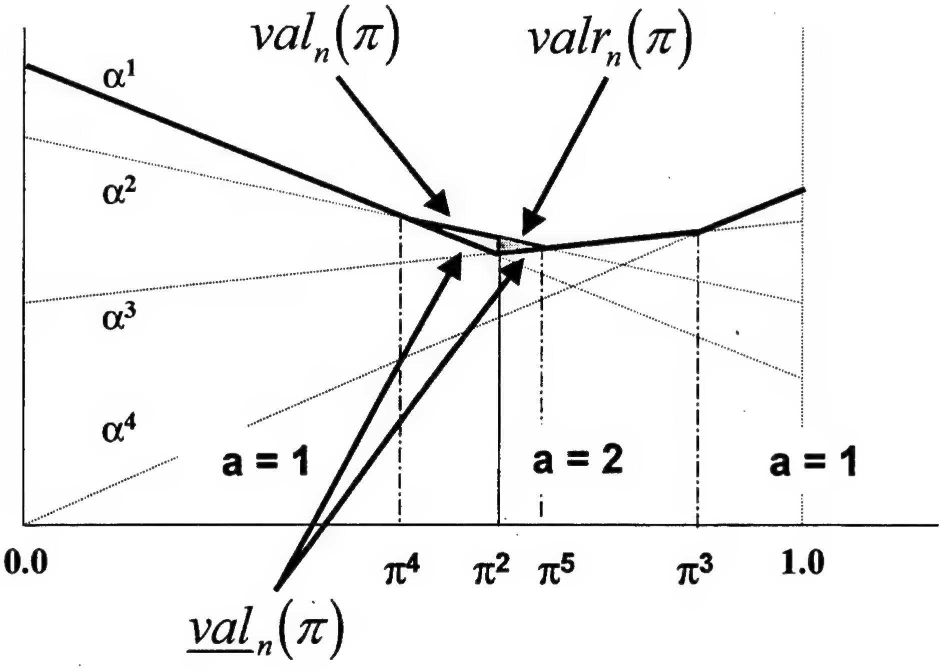

Figure 3.8: Example of the Differences Between the Linear Support Algorithm Lower Bound, the Policy Value Function, and the Optimal Value Function. In this case, the linear support algorithm does not store the vector α^2 because it does not improve the lower bound value function by more than ε. The lower-bound policy uses a = 2 in the interval [π^2, π^3], while the optimal policy uses a = 2 in the larger interval [π^4, π^3]. However, the value function resulting from the regions induced by the vectors {α^1, α^3, α^4}, $valr_n(\pi)$, is actually equal to the optimal policy $val_n(\pi)$ in the interval [π^2, 1.0]. Note that $valr_n(\pi)$ is piecewise-linear, but is not convex; also, it is discontinous due to the jump at π^2.

Now consider the example shown in Figure 3.8. This continues the example of Figure 3.7 and shows a lower bound value function that uses only 3 of the 4 possible vectors. However, the vector α^3, which covers the region associated with the omitted vector α^2, uses the same action. The vectors in $\underline{V}(n)$ partition the state space as shown in the figure. Furthermore, they induce a value function $valr_n(\pi)$ that is actually equal to the optimal value function $val_n(\pi)$ over part of the belief space; $\underline{val}_n(\pi)$ is only a lower bound. Note also that $valr_n(\pi)$ is not necessarily convex or continuous.

Suppose we have found *s* using a tolerance ε and have stored the policy using the $\underline{V}(n)$ sets. Figure 3.9 gives pseudocode for a **TREE** procedure and a recursive function to find the expected consumptions, the expected terminal reward, and the value function $valr_n(\pi)$ for a particular starting state π and time horizon *n*.

The **TREE** procedure traverses the policy tree specified by *s* starting at π(1), and accumulates the various outputs, computing the consumptions necessary to specify a column in the master LP of the decomposition. Notice, however, that the procedure does not use the stored vectors to compute the expected payoff ***val***; these vectors only determine the action to be taken in a belief state through the function ***act(s,π,t)***. As a result, the recursion computes $valr_n(\pi)$.

E. SUMMARY

This chapter has been devoted to the characteristics of POMDP solutions and the various solution algorithms available. We assert that approximate, rather the exact, methods are more appropriate for the decomposition; furthermore, we focus on the linear support algorithm as offering the most promise for the sensor-shooter problem. In addition, we offer new proofs on the error associated with the linear support algorithm, and tighten the upper bound.

```
Global $val, E(R_s), (E[Y_{si}]), (E[W_i(e,a)]), (r_e), (\lambda_i)$, policy $s$

Procedure TREE$[\pi(1), n]$
    $p \to 1$;   $\pi \to \pi(1)$;   $(E[Y_{si}]) \to (0)$;   $E(R_s) \to 0$;   $val \to 0$;
    Call RECURSE$[p, \pi, n]$;
    $val \to val + E(R_s)$;
End

Procedure RECURSE$[p, \pi, n]$
    $a \to act(s, \pi, n)$;
    if $t = 0$
        $E(R_s) \to E(R_s) + p\sum_{e \in E} r_e \pi_e$;
    else
        $(E[Y_{si}]) \to (E[Y_{si}]) + p(E[W_i(e,a)])$;
        $val \to val + \sum_{i \in I} \lambda_i E[W_i(e,a)]$;
        for each $\theta \in \Theta$
            Call RECURSE$[p \Pr(\theta|a), Tr(\pi|a,\theta), n-1]$;
        next $\theta$
    end
end
```

Figure 3.9: Procedure to Compute Expected Consumptions, Expected Terminal Rewards, and the Value Function for a Given Policy and Starting State. The procedure TREE calls a recursive function RECURSE to expand the tree for policy *s*. The policy and the POMDP data are globally accessible, and the function *act*(s,π,t) returns the action *a* used by policy *s* for state π in period *t*.

IV. IMPLEMENTING THE DECOMPOSITION

Chapters II and III establish the groundwork for a decomposition using a master LP and POMDP subproblems. However, this proposed algorithm must solve many sequences of PSPACE-complete subproblems, and the theory we have developed so far in no way guarantees that our finite algorithm is "tolerably finite." Therefore, we devote this chapter to describing our implementation and computational results of an instance of the sensor-shooter problem. In Section IV.A, we give a functional description of the problem, the formulation of the master LP, the POMDP subproblems, and the upper bounds, and describe the size of the test data. We also demonstrate the intractability of the problem using current approaches. In Section IV.B, we describe the initial implementation, which produces solutions but runs for nearly an hour on current personal computers. In Section IV.C, we present a series of refinements to accelerate the POMDP subproblems, and in Section IV.D, we present refinements to accelerate the solution of the master LPs. The combined effects of these refinements reduce the runtime for our example by over 95%.

A. DESCRIPTION AND FORMULATION OF THE SENSOR-SHOOTER PROBLEM

We introduced the sensor-shooter problem in Section I.A as one of allocating bomb-damage assessment (BDA) sensors and weapons to targets across a finite time horizon. In this section, we define in detail both the master LP and the POMDP subproblems, and show the theory developed in Chapters II and III can be applied to this problem.

1. Functional Description and Existing Modeling Approaches

The air campaign problem is as follows: we have a finite number of equal-length time periods. During each period, we have attack tactics available that we can allocate to a known set of targets. Each tactic uses a fixed number of weapons and fixed number of aircraft sorties. The choice of tactic determines the expected attrition (probability of being shot down) for the attacking aircraft. Different aircraft types can use the same types of weapons.

There are different classes of targets, and targets in each class are identical. The targets have two states (live or dead) and allocating a tactic to any target does not affect the state of any other target. Each target has a nonnegative value if it is in the dead state, and value 0 if it is alive; we assume value (utility) is additive over the targets. We also assume the attacking aircraft have no capability to determine whether the target is alive or dead; the assessment must be done with a BDA sensor (while most attack aircraft can provide assessment via imagery or pilot reports, the US military typically requires external confirmation for target status). The targets in each class all begin in the same belief state, and in this chapter, all targets are assumed to be live at the start of the time horizon. Finally, we either attack, inspect, or skip each target during a time period.

The BDA sensors are actually combinations of platforms and intelligence analysts, and each sensor in this model represents an information acquisition and processing system. The sensor gathers data about the target in some form (such as imagery, electronic emissions, or heat signature), and the analysts use data provided by the sensor to estimate the state of the target. Each of these sensor systems has known probabilities of error; that is, the probabilities of assessing a dead target as live or a live target as dead are available. We also assume (without loss of generality) that the error probabilities are independent of the target type.

In addition, each sensor type has a known response probability that is independent of the target state, and a fixed response time measured in time periods. When we look at a target, we skip attacking that target or looking at it again until the information returns. However, we know immediately if the sensor fails to respond, and we can attack the target or look at it again in the next period. In any time period, we restrict the available sensor observations to those that can respond by the last period of the time horizon.

The expected numbers of attack sorties and sensor looks are constrained for each time period. Also, the total expected consumption for each type of weapon is constrained across the time horizon, as is the total expected aircraft attrition for each aircraft type.

The objective of this problem is to maximize the expected value of the dead targets. The Air Force has used this formulation and variations of it to support weapons procurement and campaign planning for several decades, starting with the HEAVY ATTACK model (Brown, Coulter, and Washburn 1994). Other models that have been used include the Theater Attack Model (Might 1987), the Conventional Target Effectiveness Model (Cotsworth 1993), and the Conventional Forces Assessment Model (Yost 1996). These models are all mathematical programs, and are limited to the aircraft-weapon-target allocation problem. These models do not include BDA sensors explicitly; they model BDA as a noise factor that either consumes additional sorties or disguises the true state of a target for some length of time.

The only attempt to date to model sensors using one of the above models was to loosely couple the Conventional Forces Assessment Model with another LP, the Sensor-Platform Allocation Model (Rice 1997). In this study (Wilstatter and Barnes 1998), the sensor allocator generates "found" targets, the weapon allocator attacks them and in turn generates BDA requirements for the sensor allocator. However, the sensor allocator assumes perfect assessment and only models response reliability. Furthermore, the optimizations are not formally integrated.

Nonetheless, this study was of major importance in the emerging analyses of tradeoffs between weapons systems and sensors, and received attention at high levels within the US Department of Defense (Wilstatter and Barnes 1998). The importance of the topic makes at least one case for considering a decomposition such as the one we address in this dissertation.

Other attempts to study the integrated weapon-sensor problem on a smaller scale include Reed (1996) and Aviv and Kress (1997), as well as a number of simulation-based efforts. The model posed by Aviv and Kress is solved in Chapter V using the decomposition, but for now we continue with the motivating problem.

2. Master LP Formulation

The following is the formulation of the master LP, using a format that is a standard at the Naval Postgraduate School. We first list all set indices, sets, and the problem data, followed by the LP variables, the objective function, and the constraints. The dual variables associated with each constraint are given in parentheses to the right of the constraints.

- **SETS AND SET INDICES**

$i \in I$	aircraft types
$w \in W$	weapon types
$g \in G$	target types
$o \in O$	sensor types
$s \in S_g$	admissible policies for target type g
t	time period; $t = 1,2, \ldots T$

- **DATA**

TGT_g	number of targets of type g available
V_g	value of gth target type per target
$SORT_{it}$	number of sorties of ith aircraft type available in period t
$LOOK_{ot}$	number of type o sensor looks available in period t
EA_{is}	expected attrition of aircraft i using policy s
$MAXATT_i$	maximum attrition allowed for aircraft i
WPN_w	number of weapons of type w available
PD_{gs}	probability of destroying target g using policy s
EL_{ogst}	expected number of sensor o looks at target g required by policy s in period t
ES_{igst}	expected number of aircraft i sorties required by policy s against target g in period t
EW_{wgs}	expected number of type w weapons expended using policy s against target g

- **VARIABLES**

x_{gs} number of type g targets attacked using policy s

- **OBJECTIVE FUNCTION**

$$\max_{x} \sum_{g \in G} \sum_{s \in S_g} V_g \, PD_{gs} \, x_{gs} \tag{4.1}$$

- **CONSTRAINTS**

$$\sum_{g \in G} \sum_{s \in S_g} ES_{igst} \, x_{gs} \le SORT_{it}, \quad \forall\, i,t \quad (sd_{it}) \tag{4.2}$$

$$\sum_{g \in G} \sum_{s \in S_g} EW_{wgs} \, x_{gs} \le WPN_{w}, \quad \forall\, w \quad (wd_{w}) \tag{4.3}$$

$$\sum_{s \in S_g} x_{gs} = TGT_{g}, \quad \forall\, g \quad (td_{g}) \tag{4.4}$$

$$\sum_{g \in G} \sum_{s \in S_g} EL_{ogst} \, x_{gs} \le LOOK_{ot}, \quad \forall\, o,t \quad (ld_{ot}) \tag{4.5}$$

$$\sum_{g \in G} \sum_{s \in S_g} EA_{is} \, x_{gs} \le MAXATT_{i}, \quad \forall\, i \quad (ad_{i}) \tag{4.6}$$

$$x_{gs} \ge 0, \quad \forall\, g \in G, s \in S_g \tag{4.7}$$

The objective function (4.1) maximizes the expected value of dead targets. The sortie constraint (4.2) limits expected sortie consumption for each aircraft type and time period. The weapon constraint (4.3) limits expected weapon consumption across the time horizon. The policy allocation constraint (4.4) limits the assignment of policies to available targets; for feasibility, there is a null policy for each target type that consumes no resources and has a 0 probability of killing the target. The sensor constraint (4.5) constrains expected sensor looks by type and time period, and the attrition constraint (4.6) limits expected attrition by aircraft type across the time horizon.

Let $S = \bigcup_{g \in G} S_g$ be the set of all possible policies for all target types, and let $\lambda = \{sd_{it}: i \in I, t = 1,2,\ldots T\} + \{wd_w: w \in W\} + \{ld_{ot}: o \in O, t = 1,2,\ldots T\} + \{ad_i: i \in I\}$ be the set of resource dual values we pass to the subproblems. We denote the formulation above as LP(S;λ) and the optimal objective function value as $v(S)$. This problem has the form of (2.18). The expected reward is the probability of killing the target with a policy multiplied by its value, so it has the same form as the objective of (2.18). Furthermore, the constraints are all of the form of those in (2.18); the dual variables associated with target constraints are the same as the (w_j) duals defined in (2.18); the other duals match the (λ_i) duals. Note that the tactics and their consumptions are not represented explicitly in the LP, as the expected consumptions are modeled in terms of the policies.

3. POMDP Subproblem Formulation

The following indices and data are used in the POMDP:

- **SETS AND SET INDICES**

$a \in ATK_g$	set of allowable attack actions (tactics) for target g
R_g	set of terminal rewards and expected action costs for target g
p	pause action
$o \in O$	sensor look action using sensor o
n	stage, or number of time periods remaining; $n = 0,1, \ldots, T$
$e \in E$	target states {live, dead}
λ	the set of resource costs;

$$\lambda = \{sd_{it}: i \in I, t = 1,2,\ldots T\} + \{wd_w: w \in W\} + \{ld_{ot}: o \in O, t = 1,2,\ldots T\} + \{ad_i: i \in I\}$$

- **DATA**

NW_a	number of weapons required by tactic a
NS_a	number of sorties required by tactic a
PA_{ag}	probability of attrition using tactic a against a target of type g

PK_{ag}	probability of killing a target of type g using tactic a
RSP_o	response time in time periods for sensor o, a positive integer
$PRSP_o$	response probability for sensor o
δ_o	probability sensor o reports the target is live, given it is dead and the sensor responds
β_o	probability sensor o reports the target is dead, given it is live and the sensor responds
$\pi^g(1)$	initial belief that targets of type g are dead

We now specify the POMDP for a target of type g in terms of the 6-tuple $\langle E, A_g, P_g, R_g, \Theta, B \rangle$. The targets have two states, live or dead, and the sensors report directly on the state. The set of states is $E = \{live, dead\}$, and the set of observations is $\Theta = \{null, live, dead\}$, where the "null" observation results from an attack, a pause, or a sensor look that fails to report. As noted in Section IV.A.1, we assume the observation probabilities B are independent of the target type.

The set of actions for each target type consists of all possible attack tactics, sensor looks, and a "pause" action p, where pause means that we ignore the target in this time period. We also define the functions $i(a)$ and $w(a)$, which return the aircraft type i and weapon type w used for a particular attack tactic $a \in ATK_g$.

The set P_g contains the transition policies for the attack tactics, and a "null" transition probability for the look and pause actions. We assume that the targets do not change state unless attacked, so they do not die due to equipment failure, surrender, and so on. The probabilities in each P_g are defined below:

$$\left.\begin{array}{l} \Pr(\text{Dead}|a, \text{Live}) = PK_{ag}, \\ \Pr(\text{Live}|a, \text{Live}) = 1 - PK_{ag}, \\ \Pr(\text{Dead}|a, \text{Dead}) = 1 \end{array}\right\}, a \in ATK_g; \tag{4.8}$$

$$\left.\begin{array}{l}\Pr(\text{Dead}|\,o,\text{Live})=0,\\ \Pr(\text{Live}|\,o,\text{Live})=1,\\ \Pr(\text{Dead}|\,o,\text{Dead})=1\end{array}\right\},\ o\in O; \tag{4.9}$$

$$\begin{array}{l}\Pr(\text{Dead}|\,p,\text{Live})=0,\\ \Pr(\text{Live}|\,p,\text{Live})=1,\\ \Pr(\text{Dead}|\,p,\text{Dead})=1.\end{array} \tag{4.10}$$

For the set of rewards R_g, the terminal rewards are given as $r_{live} = 0$ for all targets, and $r_{dead} = V_g$. The resource costs are the same for each target state, and we show below how they are determined from the master LP dual values:

$$\begin{array}{l}(\text{pause})\ r_{pn} \equiv 0;\\ (\text{attack})\ r_{agn} \equiv NS_a\, sd_{i(a),T-n+1} + NW_a\, wd_{w(a)} + NS_a\, PA_{ag}\, ad_{i(a)}\\ \qquad\qquad n=1,2,\ldots T,\quad a\in ATK_g;\\ (\text{look})\quad r_{ogn} \equiv ld_{o,T-n+1},\quad n=1,2,\ldots T.\end{array} \tag{4.11}$$

Consequently,

$$\begin{array}{l}R_g = \{r_{live}, r_{dead}\} +\\ \qquad \{r_{pgn},\ n=0,1,\ldots T\} +\\ \qquad \{r_{agn}\!: a\in A, n=0,1,\ldots T\} +\\ \qquad \{r_{ogn}\!: o\in O, n=0,1,\ldots T\}.\end{array} \tag{4.12}$$

Note that the same attack tactic or sensor look can have different costs depending on the stage. Furthermore, the structure of the master LP guarantees that all resource costs in (4.11) are nonnegative, because the value of a dual variable corresponding to a less-than-or-equal-to constraint with a maximization objective must be nonnegative (e.g., Bazarra, Jarvis, and Sherali, 1990, p. 248).

The final set B contains the observation probabilities. For a pause or attack action, there is only a "null" observation available which does not update the belief state. Using the notation for the error probabilities defined above, we can define all the probabilities in B:

$$\begin{aligned} &\Pr(\theta = null|a,\text{Live}) = \Pr(\theta = null|a,\text{Dead}) = 1, \quad a \in ATK_g; \\ &\Pr(\theta = null|p,\text{Live}) = \Pr(\theta = null|p,\text{Dead}) = 1; \end{aligned} \tag{4.13}$$

$$\left.\begin{aligned} &\Pr(\theta = null|o,\text{Live}) = \Pr(\theta = null|o,\text{Dead}) = 1 - PRSP_o, \\ &\Pr(\theta = \text{Live}|o,\text{Live}) = PRSP_o(1-\beta_o), \\ &\Pr(\theta = \text{Live}|o,\text{Dead}) = PRSP_o\,\delta_o, \\ &\Pr(\theta = \text{Dead}|o,\text{Live}) = PRSP_o\,\beta_o, \\ &\Pr(\theta = \text{Dead}|o,\text{Dead}) = PRSP_o(1-\delta_o), \end{aligned}\right\}, \; o \in O.. \tag{4.14}$$

The belief state for this POMDP is the probability distribution over the two possible target states. However, since Pr(target is dead) = 1 - Pr(target is live), define π = Pr(target is dead), as the belief state. This makes the belief space $\Pi = \{\pi : o \le \pi \le 1\}$ for all stages.

For convenience, we show via Bayes' Theorem the updates for π for a sensor look o (4.15), the pause and attack actions (4.16), and the probabilities Pr(θ|π,a) (4.17):

$$\begin{aligned} &Tr(\pi|o,\theta = \text{Live}) = \frac{\delta_o\,\pi}{(1-\beta_o)(1-\pi)+\delta_o\,\pi}, \\ &Tr(\pi|o,\theta = \text{Dead}) = \frac{(1-\delta_o)\pi}{\beta_o(1-\pi)+(1-\delta_o)\pi}, \end{aligned} \tag{4.15}$$

$$\begin{aligned} &Tr(\pi|a,\theta = \text{null}) = \pi + PK_{ag}(1-\pi), \quad a \in ATK_g, \\ &Tr(\pi|o,\theta = \text{null}) = \pi, \qquad o \in O, \\ &Tr(\pi|p,\theta = \text{null}) = \pi; \end{aligned} \tag{4.16}$$

$$\begin{aligned}
&\Pr(\theta = null | \pi, a) = 1, \quad a \in ATK_g, \\
&\Pr(\theta = null | \pi, p) = 1, \\
&\left.\begin{aligned}
&\Pr(\theta = null | \pi, o) = 1 - PRSP_o, \\
&\Pr(\theta = \text{Live} | \pi, o) = PRSP_o\left[(1-\beta_o)(1-\pi) + \delta_o \pi\right], \\
&\Pr(\theta = \text{Dead} | \pi, o) = PRSP_o\left[\beta_o(1-\pi) + (1-\delta_o)\pi\right]
\end{aligned}\right\}, o \in O.
\end{aligned} \tag{4.17}$$

The POMDP is now completely specified as in Chapter II. However, we present the DP recursion in a form that better illustrates the underlying model. For a target in a particular class g, the DP recursion $DP^g(\lambda)$ is

$$val_0^g(\pi) = V_g \pi, \tag{4.18}$$

$$val_n^g(\pi) =$$
$$\max\begin{cases}
(\text{pause}) \quad val_{n-1}^g(\pi), \\
(\text{attack}) \quad \max\limits_{a \in ATK_g}\left\{-r_{agn} + val_{n-1}^g\left(\pi + PK_{ag}[1-\pi]\right)\right\}, \\
(\text{look}) \quad \max\limits_{o:n>RSP_o}\left\{\begin{aligned}&-r_{ogn} + (1-PRSP_o)val_{n-1}^g(\pi) + \\ &\sum_{\theta\in\Theta}\Pr(\theta|\pi,o)val_{n-RSP_o}^g\left(Tr[\pi|o,\theta]\right)\end{aligned}\right\}
\end{cases} \tag{4.19}$$
$$n = 1, 2, \ldots T.$$

The fact that we are using value functions from other than the n-1st stage in (4.19) has no effect on the piecewise-linearity and convexity of the value function for stage n. The nth-stage value function can still be written as the maximum of a set of linear functions, and Theorem 3.1 still applies. This formulation also hints at the wide variety of models that are possible in the POMDP subproblem.

4. POMDP Structural Results

As we noted in Chapter III, finding structural results for POMDPs is difficult. In the case of the DP recursion (4.18) and (4.19), we can get some insights, but these do not substantially alter our approach.

Theorem 4.1: $val_n^g(\pi) \le val_{n+1}^g(\pi)$ **for all** $n = 0,1,\dots T-1$.

Proof: In stage n+1, we can earn at least the expected payoff from the nth stage by pausing for all belief states. ■

We state the next two theorems without proof:

Theorem 4.2: If the expected cost of taking an action is more than the value of the target, that action is never used and does not need to be considered, that is,

$$\begin{aligned} r_{agn} > V_g &\Rightarrow \text{tactic } a \text{ is not used any stage } n,\ a \in ATK_g; \\ r_{ogn} > V_g &\Rightarrow \text{observation } o \text{ is not used any stage } n. \end{aligned} \qquad (4.20)\ ■$$

We can also remove any "dominated" actions:

Theorem 4.3: If an attack action has a higher expected cost and a lower probability of kill than another attack action, the former attack action is never used. If a sensor look has a higher cost, higher error rates, and a longer response time than another sensor, the former sensor look is never used. ■

We also have a stronger domination test that relies on the following result:

Theorem 4.4 (MacQueen, 1967): Let $val_n^g(\pi)$ **be as in (4.18) and (4.19), and suppose we have two functions** $vu_n^g(\pi)$ **and** $vl_n^g(\pi)$ **such that** $vl_n^g(\pi) \le val_n^g(\pi) \le vu_n^g(\pi)\ \forall \pi \in [0,1]$**. Then if** $-r_{agn} + vu_n^g(\pi) \le vl_n^g(\pi)\ \forall \pi \in [0,1]$**, then action** a **is never used in stage** n**; similarly, if** $-r_{ogn} + vu_n^g(\pi) \le vl_n^g(\pi)\ \forall \pi \in [0,1]$**, then action** o **is never used in stage** n**.** ■

Corollary 4.5: If $-r_{agn} + V_g \le val_{n-1}^{g}(\pi)\ \forall \pi \in [0,1]$, then action *a* is never used in stage *n*; also, if $-r_{ogn} + V_g \le val_{n-1}^{g}(\pi)\ \forall \pi \in [0,1]$, then action *o* is never used in stage *n*.

Proof: Let $vu_n^g(\pi) = V_g$ and $vl_n^g(\pi) = val_{n-1}^{g}(\pi)$. By Theorem 4.1, we cannot earn more than the value of a target, so it is clearly an upper bound; also, the pause action ensures we can always earn the payoff in the *n*-1st stage. The result follows by applying Theorem 4.4. ■

We can also know the value function increases with the belief state:

Theorem 4.6: $val_n^g(\pi)$ is nondecreasing in π for all target types *g* and *n* = 0,1, ... *T*.

Proof: The theorem is true for *n* = 0 by the definition of the value function (4.18). Now, assume $val_{n-1}^{g}(\pi)$ is increasing for an arbitrary stage *n*-1. For any action, the transformations given in (4.15) and (4.16) are increasing in π, so $val_{n-1}^{g}(Tr[\pi|a,\theta])$ is increasing in π. Any choice of action adds a constant or multiplies $val_{n-1}^{g}(Tr[\pi|a,\theta])$ by a nonnegative number to compute $val_n^g(\pi)$, so $val_n^g(\pi)$ is also increasing in π. ■

These structural results above are not tremendously useful. Theorem 4.2, Theorem 4.3 and Corollary 4.5 do save some work in the linear support algorithm because they can be used to reduce the size of the action set. The other results characterize the solutions, but these are largely confirmations that the model makes functional sense.

However, one structural consequence of the model allows us to quickly determine a lower bound for the solution to the *n*th stage. Since pausing in the *n*th stage merely earns the expected reward in the *n*-1st stage for any belief state, we can immediately set $\underline{val}_n(\pi) = val_{n-1}(\pi)\quad \forall \pi \in [0,1]$, and the initial set of α-vectors *V*(*n*) = *V*(*n*-1). Now, whether or not this makes solving the POMDP any easier depends on the solution method. We exploit this property in our implementation of the linear support algorithm (Section IV.B.2).

There is one other issue associated with the structure of the solutions. In reality, there is no sense in expending resources on a target known to be dead (a belief state of 1). However, it is possible that certain resource constraints are not binding in the solution of the master LP. In this case, these resources have marginal costs of 0 and the POMDP subproblems see actions using them as free. As a result, there can be a tie for the choice of action in the DP recursion for the belief state $\pi = 1$. Following reality, our tie-breaking rule is that the optimal action for the belief state $\pi = 1$ is to pause in all cases.

This POMDP resembles the machine-maintenance problem studied by Monahan (1980), whose action set consists of "continue operating the machine," "stop the machine," or "inspect the machine." The machine has two states (good or bad) and Monahan (1980, p. 1327) was able to prove that the "stop" region was convex. However, our pause action is different from a stopping action, in that we may later attack the target and change its state in later stages. We have generated examples where the pause region is not convex, so Monahan's convexity result does not hold.

5. Dimensionality of Test Data

To test the decomposition, we use a notional version of a US Air Force campaign database. The data includes 9 attack aircraft types, 42 weapon types, 65 target types, and 10 sensor types. There are 5,203 total tactics; the target types have between 37 and 139 applicable attack tactics, with an average of 80 tactics each. There are 6,316 total targets, with an average of 97 targets in each class. The values of the targets in each class range from 1 to 74, and the total available target value is 84,170. We assume that each of the 10 sensors can assess any of the target types.

We use a time horizon of 9 periods (3 operating periods per day for 3 days), which is a typical wartime planning horizon. The sorties and sensor looks available are constrained in each time period, but the weapons and the attrition are constrained over the entire time horizon.

As noted in Chapter 1, a straightforward assault on this problem does not work. Using linear programming results in a formulation with an intractable number of columns; (1.4) shows that the number of policies possible in a 6-period problem for a target with possible 80 attack tactics and 10 sensor looks is 3.03×10^{92}. Therefore, we cannot even handle a problem with one target type. Similarly, we cannot solve this problem as one large POMDP either. There are 2^{6316} possible target states, which is beyond the capability of any known POMDP algorithm.

Yet, the master LP is very simple, having only 287 constraint rows (9 x 9 = 81 sortie constraints, 42 weapon constraints, 65 target constraints, 10 x 9 = 90 sensor look constraints, and 9 attrition constraints). Similarly, the POMDP for each target type is of the kind that is considered trivial in the literature, as there are only two target states and two possible observations.

We again comment here on the usefulness of the decomposition. Lovejoy (1991, p. 62-63) speculates that POMDPs have not been used much because they are restricted to problems with small numbers of states and observations, and he comments that a time horizon of $T = 100$ steps seems to be the norm for any real problem. However, many POMDPs (and MDPs) require the extra states to account for constraints that we handle through the dual prices provided by the master LP. Also, the long time horizon seems to be a common refrain in the MDP literature, where infinite-horizon, stationary policies are the preferred solution. In our problem, the last thing we want are stationary policies, as combat is not a steady-state process.

6. The Overall Decomposition for the Sensor-Shooter Problem

At this point, it is worthwhile to revisit Figure 1.2 for the overall decomposition. Figure 4.1 illustrates the overall decomposition scheme, relabeled for the sensor-shooter problem.

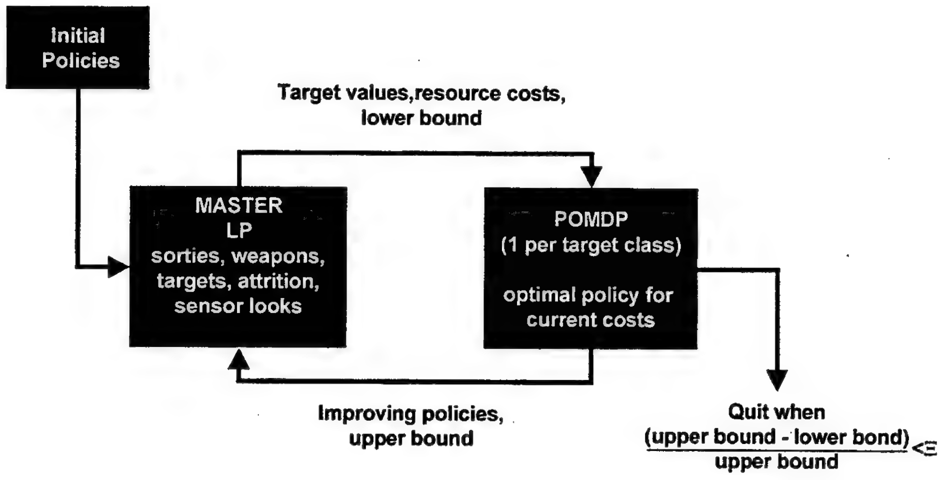
Initial Policies
Target values,resource costs, lower bound
MASTER LP sorties, weapons, targets, attrition, sensor looks
POMDP (1 per target class)
optimal policy for current costs
Improving policies, upper bound
Quit when (upper bound - lower bond) / upper bound <Ξ

Figure 4.1: Basic Decomposition Algorithm for the Motivating Problem. The master LP computes a lower bound for the current set of policies, determines marginal resource costs for sorties, weapons, attrition, and sensor looks, and passes them to the POMDPs. The POMDPs use those costs to determine improving policies and compute an upper bound. The algorithm ends when the relative gap between the two bounds is less than some specified tolerance Ξ.

LP(S;λ), the master LP, is given by (4.1) and (4.2)-(4.7). $DP^g(\lambda)$, the DP recursion used to solve the POMDP subproblems is given by (4.18) and (4.19). The only thing left to specify for the algorithm is the upper bound computation LPU(S;λ):

$$LPU(S;\lambda) =$$

$$\max_x \left\{ \begin{array}{l} \sum_{g\in G}\sum_{s\in S_g} V_g\, PD_{gs}\, x_{gs} + \sum_{i\in I}\sum_t sd_{it}\left(SORT_{it} - \sum_{g\in G}\sum_{s\in S_g} ES_{igst}\, x_{gs}\right) + \\ \sum_{w\in W} wd_w\left(WPN_w - \sum_{g\in G}\sum_{s\in S_g} EW_{wgs}\, x_{gs}\right) + \sum_{o\in O}\sum_t ld_{ot}\left(LOOK_{ot} - \sum_{g\in G}\sum_{s\in S_g} EL_{ogst}\, x_{gs}\right) + \\ \sum_{i\in I} ad_i\left(MAXATT_i - \sum_{g\in G}\sum_{s\in S_g} EA_{is}\, x_{gs}\right) \end{array} \right\}.$$

$$\begin{array}{lll} st & \sum_{s\in S_g} x_{gs} = TGT_g & \forall\, g\in G \\ & x_{gs} \geq 0 & \forall\, g\in G, s\in S_g\,. \end{array} \qquad (4.21)$$

LPU(S;λ) decomposes into |G| separate optimizations, one for each target class. Following (2.20), the solution for each target class optimization is

$$u_g\left(S_g;\lambda\right) \equiv val_T^g\left[\pi^g(1),\lambda\right]. \qquad (4.22)$$

And, following (2.21), the upper bound u(S;λ) is

$$u(S;\lambda) \equiv \left\{ \begin{array}{l} \sum_{i\in I}\sum_t sd_{it}\, SORT_{it} + \sum_{w\in W} wd_w\, WPN_w + \\ \sum_{o\in O}\sum_t ld_{ot}\, LOOK_{ot} + \sum_{i\in I} ad_i\, MAXATT_i + \\ \sum_{g\in G} TGT_g\, u_g\left(S_g;\lambda\right) \end{array} \right\}. \qquad (4.23)$$

With the upper bound defined, we can describe the entire algorithm for the motivating problem. Let $T^{g,k}$ be the set of policies (columns) available for the master LP to use for target g in iteration k, and let $T^k = \bigcup_{g\in G} T^{g,k}$ be the set of all possible columns in iteration k. We defer the issue of how to generate the initial policies to Section IV.B, so assume we have T^1 and can begin the decomposition. Following Figure 4.1 in general and the algorithm in Figure 2.4 in particular, we solve LP(T^1;λ^1) for the lower bound $v(T^1)$ and the resource prices λ^1. We then solve the

POMDP subproblems $DP^g(\lambda)$ for all target classes g, and compute the upper bound $u(S,\lambda^1)$. If $u(S;\lambda_1) - v(T^1) > \Xi$, we add new columns to T^1 to create T^2, and continue.

It is helpful to check the columns prior to adding them. For a column to price favorably, $u_g(S;\lambda^k) - td_g > 0$. As noted in Section II.C, there is no guarantee that the POMDP will produce an improving column; indeed, the resource prices may be such that it is not worth the expenditure to attack a target of small value. While these columns may be added without affecting the master LP, we discard them in our implementation.

7. Generating Initial Policies

Referring back to Figure 4.1, we must start the algorithm using either the master LP or the subproblems. While the flow diagram begins with a set of initial policies, we could just as easily start the algorithm in the subproblems with an initial set of resource prices. The choice of how to start the algorithm is model-dependent. In a situation where the dual resource prices may have an economic interpretation, it may be easy to estimate an initial set of costs and use them to generate policies from the POMDPs.

This is not the case in the sensor-shooter problem. While there is plenty of expert judgement available in the Air Force campaign-planning community on relative merits of different types of sorties, weapons, and sensor looks, translating those opinions into a set of numbers is not easy. Also, it makes sense from a modeling point of view to start with *existing* policies used by campaign planners. In addition to increasing model credibility, this allows us to compare the recommendations provided by the optimization and the heuristic solutions developed from doctrine and experience.

Our heuristic policy generator is very simple. For each type of target, we compute the best 3 "single-shot" policies for each time period. These policies specify a single attack with a single tactic in one period, and pauses in all other periods.

Next, we generate the 3 best "shoot-look-shoot" policies for each target type and each applicable time interval. These policies pick a tactic and a sensor look. They first attack with the tactic, look with the sensor in the next period, wait for the sensor to respond, and attack using the same tactic if the sensor responds and indicates the target is live.

Finally, we find the best single tactic for each attack aircraft sortie and each weapon, and add an associated single-shot policy for each of them in each time period. We also find any sensors that were not used in any of the shoot-look-shoot policies, and add corresponding shoot-look-shoot policies for them in all applicable time intervals.

We generate this last group of policies to make sure the master LP can generate dual costs for each resource. If a certain resource isn't used by any policy, the LP cannot compute dual information on the value of the resource and the POMDP sees it as free. Now, any resource constraint that doesn't bind the master LP solution has a dual price of 0, so the resource actually is free. However, if no policy uses the resource, the dual price is not necessarily 0; it is unknown.

Employing this heuristic for the test data set generates 2,214 policies, which is small compared to the number of policies possible, but a good approximation of current campaign planning practices.

B. INITIAL IMPLEMENTATION OF THE SENSOR-SHOOTER MODEL

1. Software and Hardware

The decomposition is written in Visual Basic 5.0 (Microsoft, 1997) compiled to native code, and the routines that solve the POMDP subproblems are all written in Visual Basic. We report computation times using a Dell XPS333 equipped with a 333 MHz Pentium-II processor, 196 MB of memory, and running Windows NT 4.0.

The linear programs are solved using the CPLEX 5.0 Callable Library (ILOG, 1997). We discuss an experiment using interior-point methods for solving the LP's in Section IV.D.2, but we used the simplex method in all other cases.

The decomposition for the sensor-shooter problem is not memory-intensive. We have solved the same problem on personal computers and laptops with as little as 32 MB of memory with no difficulties.

2. Implementation of the Linear Support Algorithm

It is worth discussing how we implemented the linear support algorithm, as our code is tailored specifically to the sensor-shooter POMDP and does not generalize to other models.

The data structure for the POMDP policy is a T+1 x 1 array of records, and each of these records is itself an array of α-vector records for the particular time period. We store the components of the vectors and the associated actions, and, since the α-vectors are just line segments in R^2, we store the endpoints of the interval of the belief space [0,1] covered by each α-vector. The stored action associated with each α-vector provides the map from the belief state to the set of actions.

For any stage n, the DP recursion has to evaluate $val_{n-1}^{g}(\pi)$ for many different belief states. To accelerate these evaluations, we store the α-vector arrays in order of the left endpoints of their intervals, and use a binary search to find the interval of the belief space that contains π and the α-vector associated with that interval.

We begin the algorithm by storing the single α-vector for the stage $n = 0$, which is $\alpha^1(0) = \left(0, V_g\right)$. For any succeeding stage, we proceed as follows. We first use Theorem 4.2, Theorem 4.3 and Corollary 4.5 to reduce the number of tactics. Furthermore, we can immediately compute an initial value function using the previous stage by setting $val_n^g(\pi) = val_{n-1}^g(\pi)$; this corresponds to pausing for all belief states. In our implementation, this means that we copy the array of α-vectors for the n-1st stage to the array for the nth stage, and change all the associated actions to p.

We then initialize a "point list" for all endpoints of the intervals for all the α-vectors for the n-1st stage. The records in this list contain the point in the belief space and the indices of the vectors that have interval boundaries at this point. As outlined in Section III.D.1, we begin by removing a point from this list and find the optimal action at that point using (4.16). If the improvement in the value function is less than ε, we store the improvement as the maximum error ε_n so we can use Corollary 3.5 to compute upper bounds.

On the other hand, if the improvement at point π is greater than the tolerance ε, we compute components of the new vector $\alpha'(n)$. To give an example of this computation, suppose attack action a' is optimal at point π. Let $l(\pi, a', \theta)$ be the index of the vector that maximizes the value function when written in the form of (3.3). The following vectors and matricies are defined by the problem data:

$$R_{a'gn} = \begin{pmatrix} r_{a'gn} \\ r_{a'gn} \end{pmatrix}; \tag{4.24}$$

$$P_g^{a'} = \begin{bmatrix} 1 - PK_{a'g} & PK_{a'g} \\ 0 & 1 \end{bmatrix}; \tag{4.25}$$

$$\Theta^{a'}_{\text{Null}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Theta^{a'}_{\text{Live}} = \Theta^{a'}_{\text{Dead}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{4.26}$$

Then via (3.6), the components of the new vector are

$$\begin{aligned} \alpha'(n) &= \sum_{\theta \in \Theta} \left[-\frac{R_{a'gn}}{|\Theta|} + P_g^{a'} \cdot \Theta_\theta^{a'} \cdot \alpha^{l(\pi,a',\text{null})}(n-1) \right] \\ &= -R_{a'gn} + \begin{pmatrix} \left(1 - PK_{a'g}\right) \alpha_1^{l(\pi,a',\text{null})} + PK_{a'g}\, \alpha_2^{l(\pi,a',\text{null})} \\ \alpha_2^{l(\pi,a',\text{null})} \end{pmatrix} \\ &= \begin{pmatrix} -r_{a'gn} + \left(1 - PK_{a'g}\right) \alpha_1^{l(\pi,a',\text{null})} + PK_{a'g}\, \alpha_2^{l(\pi,a',\text{null})} \\ -r_{a'gn} + \alpha_2^{l(\pi,a',\text{null})} \end{pmatrix} \\ &\equiv \begin{pmatrix} \alpha_1' \\ \alpha_2' \end{pmatrix}. \end{aligned} \tag{4.27}$$

Then, the value function at the point π is computed as

$$val_n(\pi) = \begin{pmatrix} 1-\pi & \pi \end{pmatrix} \begin{pmatrix} \alpha_1' \\ \alpha_2' \end{pmatrix} = \alpha_1' + (\alpha_2' - \alpha_1')\pi . \tag{4.28}$$

The latter form is just a line in R^2, and we use this form to determine the interval the new vector covers. Since we store the α-vectors in order of their intervals, we can quickly search the array of vectors starting from the current point (which must either be the intersection of two existing vectors, the point 0, or the point 1) to determine if the new vector dominates any of the existing vectors. That is, if $\alpha_1' + (\alpha_2' - \alpha_1')\pi > \alpha_1^k + (\alpha_2^k - \alpha_1^k)\pi$ for all π in the interval covered by the *k*th vector, the latter vector is dominated.

We remove the endpoints of any dominated α-vectors from the point list, and delete the dominated vectors from the vector array for the *n*th stage; note that since every new vector is optimal at some point, only the original vectors can be dominated. If the domination test results in a tie at the point π = 1, we define an α-vector whose interval is the point [1,1] and corresponds to a pause action to enforce our rule of only pausing for dead targets. Then, we compute the interval for the new vector and add those points to the list.

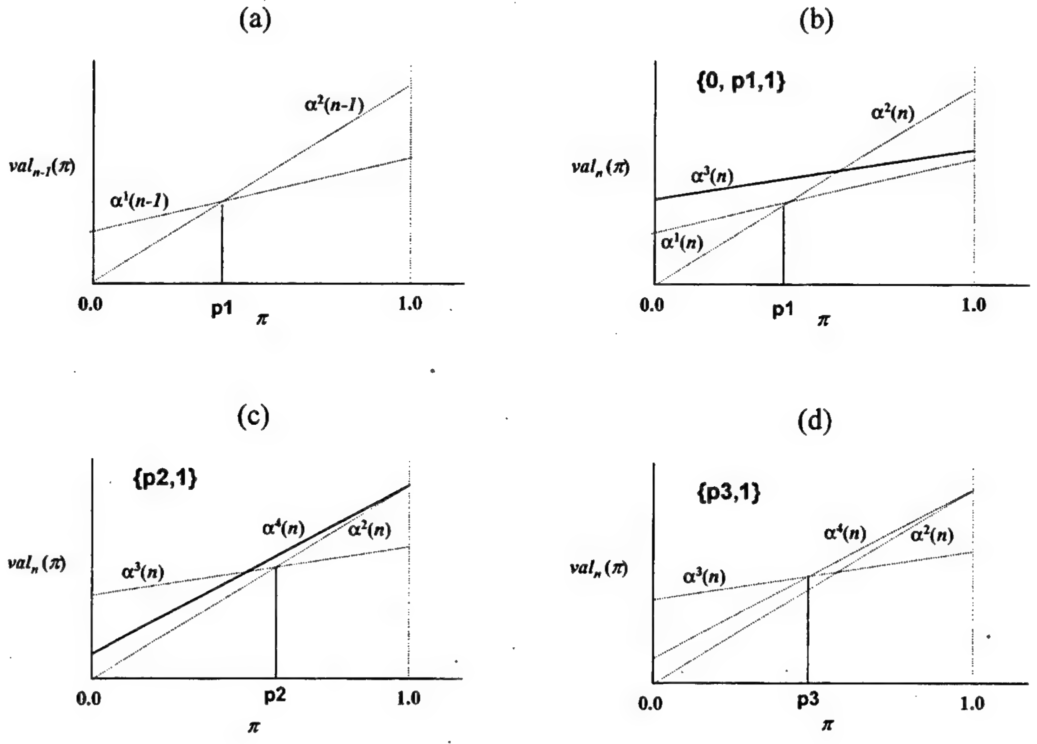

Figure 4.2: Example of the Linear Support Algorithm for the Sensor-Shooter Problem. In (a), the value function for the nth stage is initialized using the α-vectors from stage n-1, and changing their actions to pause. In (b), the point list is initialized at {0,p1,1}. Testing at π = 0 uncovers the vector α^3(n). This vector dominates α^1(n), removes point p1 from the point list, and adds point p2. In (c), we find the new vector α^4(n) at π = p2. This vector dominates α^2(n) at every point but π = 1. In (d), there is no further improvement, and the value function is optimal. By convention, the interval associated with α^2(n), which is the pause action, only consists of the point π = 1.

We continue generating and testing vectors, and updating ε_n until the point list is empty. The value function is now optimal for all belief states to a tolerance of ε_n. The remaining vectors, intervals, and actions completely describe both the value function and the policy for this stage. Figure 4.2 shows an example of the algorithm, with an initial set of α-vectors and an initial point list. The successive steps of the example show the initial value function, the computation of a new vector, the removal of dominated vectors, and the updates to the point list.

Even when using the exact version of this algorithm, we set ε at 10^{-9} to avoid generating nearly identical vectors and creating very small intervals. In our instance of the sensor-shooter problem, we have seen cases where the stored policy for a single stage consists of over 300 α-vectors, many of which cover intervals smaller than 0.001 wide.

3. Initial Computational Results

Figure 4.3 shows the progress of the decomposition algorithm by iteration and by elapsed time for the sample data. In this test, we use a numerical tolerance of $\varepsilon = 10^{-9}$ in the POMDP subproblems, and initialize the master LP using 2,214 policies generated by the heuristic. We set the gap tolerance Ξ to 0.001 (0.1%); that is, the decomposition stops when

$$\frac{u(S;\lambda^k)-v(T^k)}{u(S;\lambda^k)} < 0.001.$$

The decomposition takes 2,608 seconds and 98 master LP-subproblem iterations to solve the problem. The POMDP subproblems generate 4,574 additional policies in addition to the 2,214 heuristically-generated policies, a very small fraction of the total number of possible policies.

The final objective function value of 61,137.98 is noteworthy, as it is over 3 times larger than the value of 19,828.01 achieved using only the policies from the heuristic. This increase has significant operational implications, and also shows that solving the POMDPs to find improving policies is worthwhile for this model.

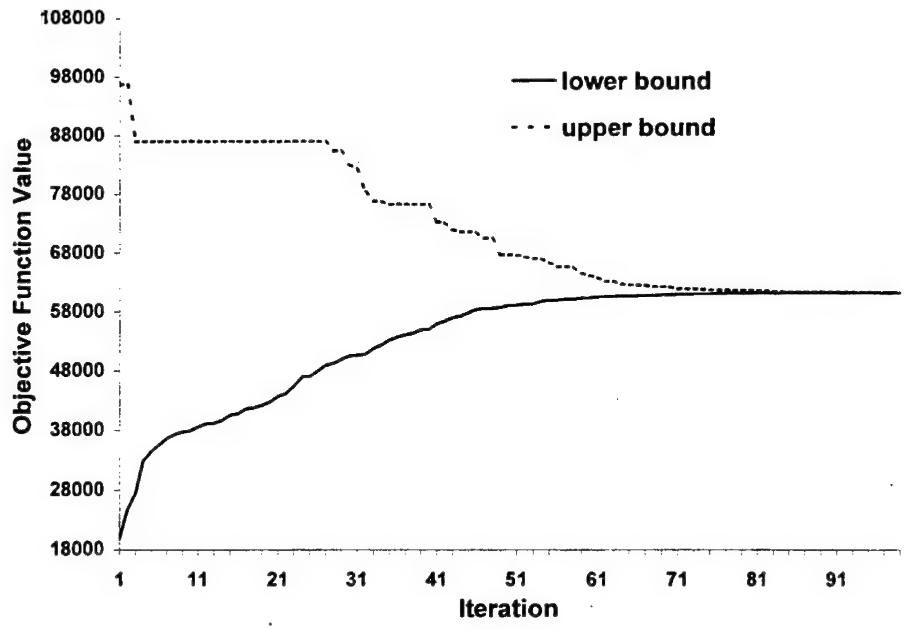

Figure 4.3: Results of the Decomposition with POMDP Solutions Exact to a Numerical Tolerance. The decomposition is initialized with 2,214 policies generated by a heuristic, and takes 98 iterations to solve to within a 0.1% decomposition gap. The POMDP subproblems generate a total of 4,507 policies, and the final objective function value is nearly triple the objective function value achieved using only the heuristically-generated policies.

Figure 4.4 shows the progress of the lower bound as function of run time. This plot shows that the decomposition exhibits the slow tail convergence typical of column-generation algorithms, with 53% of the solution time devoted to reducing the decomposition gap from 5% to 0.1%.

While finding a solution for a problem that is intractable with current methods is encouraging, the time required to solve the problem on a relatively fast PC is disappointing. Solving the master LP's only requires 93 seconds of the total solution time, while the POMDPs consume 2,483 seconds (the remaining 32 seconds are overhead in the Visual Basic code). Since over 95% of the computing time is consumed solving POMDPs, they are the obvious target for improvement.

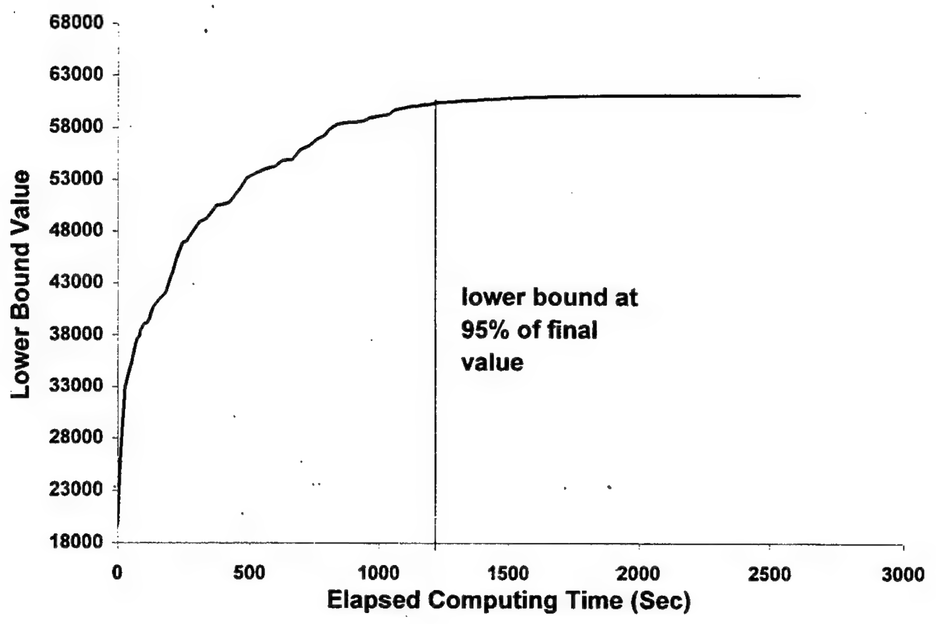

Figure 4.4: Lower Bounds as a Function of Elapsed Time for the Decomposition with Exact POMDP Solutions. This chart shows that the decomposition has slow tail convergence that is typical of column-generation algorithms. 53% of the total solution time is spent achieving the final 5% of improvement.

To provide a basis of comparison for the improvements we discuss in the next 2 sections, note that the POMDP subproblems can be solved in parallel. With perfect parallel computing, the subproblem solution time would be reduced to the maximum time required for any subproblem. For this set of data, the maximum subproblem time for any target class is 199 seconds, so we could in theory reduce the total time to 93 + 199 + 32 = 324 seconds by simply adding more processors.

C. ACCELERATING THE POMDP SUBPROBLEMS

Examining the policies generated by the POMDPs in the solution shown in Section IV.B.3 reveals that some are extraordinarily complicated; one contains a total of 1,743 α-vectors.

Furthermore, the decomposition generates a large number of policies (4,574) relative to the number of constraints in the master LP (281).

This behavior is similar to that reported by Cassandra, Littman, and Zhang (1997, p. 59) in their experiences testing exact POMDP algorithms. They report that 95% of the computing time in the algorithms is spent solving LPs, which are either doing dominance testing on α-vectors or searching for new points to test in the belief space. As we show in Table 3.2, some of these test problems have a large number of observations and long time horizons. This implies that there are a great number of nearly parallel α-vectors covering very small regions in each optimal policy. There are no published analyses of the numerical behavior of the LPs solved as subproblems in the various exact POMDP algorithms, but we speculate that a simplex code trying to solve such a problem would do many nearly-degenerate pivots and would encounter many nearly singular basis inverses. The net result would be long computing times.

The linear support algorithm is particularly sensitive to the number of α-vectors in the policy, which is why it could not solve any of Cassandra, Littman, and Zhang's test problems. Linear growth in the number α-vectors results in exponential growth in the number of points the algorithm must search, so insisting on exact solutions results in computational intractability.

Yet, retaining vectors that provide insignificant improvement and cover tiny regions makes little sense for our decomposition (or any other POMDP, for that matter). In the rest of this section, we offer compelling empirical evidence that the POMDPs should be solved with loose` tolerances, and that such a scheme can work without affecting the quality of the overall solution.

1. Effects of POMDP Tolerance Settings on Decomposition Bounds

We proved in Theorem 3.4 that for a T-period POMDP, the maximum error in the value function when using tolerance ε in the linear support algorithm is $T\varepsilon$. Using this result in (4.22) means that

$$u_g\left(S_g;\lambda\right) \le val_T^g\left[\pi^g(1),\lambda\right] + T\varepsilon. \qquad (4.29)$$

Therefore, the true upper bound obeys the following inequality:

$$u(S;\lambda) \le \left\{ \begin{array}{l} \sum_{i\in I}\sum_t sd_{it}\, SORT_{it} + \sum_{w\in W} wd_w\, WPN_w + \\ \sum_{o\in O}\sum_t ld_{ot}\, LOOK_{ot} + \sum_{i\in I} ad_i\, MAXATT_i + \\ \sum_{g\in G} TGT_g\, val_T^g\left[\pi(1),\lambda\right] \end{array} \right\} + T\varepsilon \sum_{g\in G} TGT_g\,. \qquad (4.30)$$

In our initial solution of the sensor-shooter example, we were willing to tolerate a decomposition gap of $\Xi = 0.001$. When the algorithm terminated, the lower bound objective function value was 61,137.98, and the upper bound value was 61,193.65. Setting $\varepsilon = 10^{-9}$ makes the upper bound accurate to within $6{,}316 \times 9 \times 10^{-9} = 0.0000568$. This is 6 orders of magnitude smaller than the absolute decomposition gap of 55.67, so we can certainly afford to loosen the POMDP tolerance.

When we solve the POMDPs with a tolerance other than 0, the only change required in the decomposition is to modify (4.22) using Corollary 3.5. Let $eps_t^g(\varepsilon)$ be the maximum error computed in period t for target g as in (3.29) with a tolerance setting of ε. Then, we substitute the following upper bound for $u_g(S;\lambda)$ in (4.29):

$$\overline{u}_g\left(S_g;\lambda\right) \equiv val_T^g\left[\pi^g(1),\lambda\right] + \sum_{t=1}^{T} eps_t^g(\varepsilon). \qquad (4.31)$$

Using (4.31), with ε set to 10^{-4}, the lower bound objective function value of 61,137.56 is nearly identical to the previous solution. Interestingly enough, the upper bound is tighter at 61,185.16. The second solution produces slightly fewer columns (4,548 versus 4,574) and uses the same number of iterations.

Nevertheless, the solution time is cut by over half, from 2,608 seconds to 1,179 seconds. The reduction is nearly all in the POMDP subproblems; their total time is reduced from 2,483 seconds to 1,062 seconds. Furthermore, the policies generated have many fewer vectors. The most complex policy in the second run contains only 516 vectors, compared to the 1,743-vector policy generated in the first run.

When we run the decomposition with ε = 10^{-2}, we actually get a better lower bound. After 100 iterations, the algorithm does not meet the decomposition gap tolerance, but produces a lower bound value of 61,139.97, which is higher than the first two runs (the upper bound was 61,381.52). The most complicated policy contains only 131 vectors, and the solution time drops to 384 seconds.

Table 4.1 summarizes these statistics for various POMDP tolerances. The POMDP error *errp*(ε) is computed as follows from the POMDPs solved in the final iteration of the decomposition:

$$errp(\varepsilon) \equiv \sum_{g \in G} \left(TGT_g \sum_{t=1}^{T} eps_t^g(\varepsilon) \right). \tag{4.32}$$

We compare this to the maximum POMDP error,

$$maxerrp(\varepsilon) \equiv T\varepsilon \sum_{g \in G} TGT_g . \tag{4.33}$$

We limit each of the decomposition runs in Table 4.1 to 100 iterations. We also terminate the decomposition if none of the POMDPs can generate an improving policy. The latter only occurs with the POMDP tolerance set to 0.5, which is very coarse for this data set; recall that some target types only have a value of 1.

POMDP Tolerance	Objective Bounds lower	Objective Bounds upper	Solution Times (sec) POMDPs	Solution Times (sec) LPs	Solution Times (sec) Total
0.000000001	61137.98	61193.65	2483	93	2608
0.0001	61137.56	61185.16	1062	86	1179
0.01	61139.97	61381.52	264	88	384
0.1	61052.03	63545.74	133	43	204
0.5	59935.62	71697.31	32	9	53

POMDP Tolerance	POMDP error	max POMDP error	# vectors in max policy	policies generated	iterations
0.000000001	0	0.000056844	1743	4574	98
0.0001	1.73	5.68	516	4548	98
0.01	232.99	568.44	135	4487	>100
0.1	2493.71	5684.40	65	2810	>100
0.5	11761.69	28422.00	31	1326	46*

Table 4.1: Solution Statistics for Various POMDP Error Tolerances. These two tables show various solution statistics for the sensor-shooter example. Tightening the POMDP tolerances gives insignificant improvements in the solution lower bound while significantly increasing runtime. Also, the actual POMDP error (4.32) is less than half of the maximum possible (4.33) for each tolerance setting. The decomposition is run for a maximum of 100 iterations in each case; the tolerance = 0.5 case terminates at 46 iterations because it cannot generate any more improving policies.

While these are empirical results for a single instance of a single model, they yield some interesting insights. First of all, solving the POMDPs exactly is a bad idea. We can get just as good a result in less than half the computing time with a relaxed tolerance setting. Also, the actual POMDP error is less than half of the maximum possible error, and the complexity of the policies decreases dramatically as the POMDP tolerance is relaxed.

Figure 4.5 shows a POMDP for a single target type with a single set of resource costs run at various tolerance settings. This figure also illustrates the tradeoff between the POMDP tolerance and the accuracy of the value function. We use two y-axis scales in this figure to show at what tolerance level the decrease in the value function "crosses" the number of vectors in the policy. The decrease in the value function is the decrease from $val_9(0)$; that is, for a target known to be live at the start of the 9-period horizon.

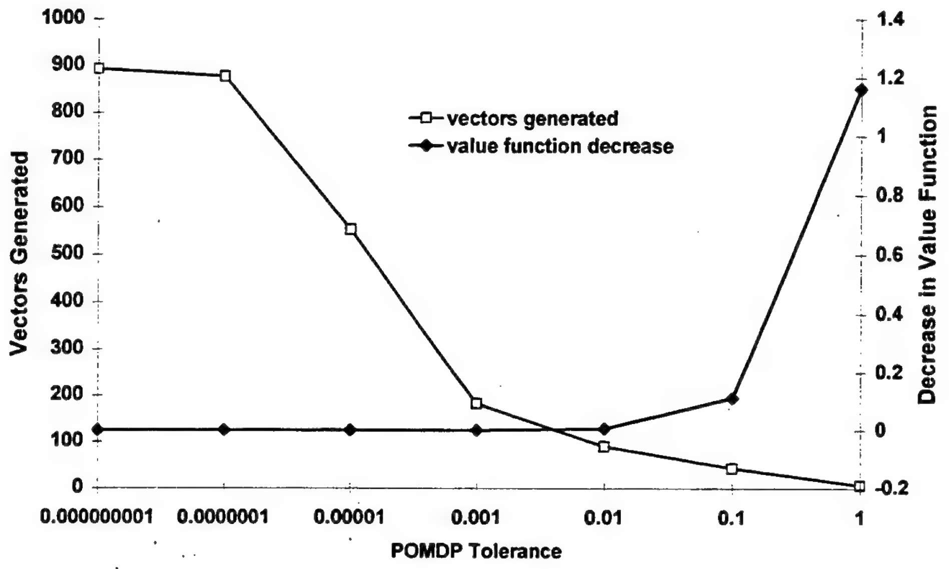

Figure 4.5: Vectors Generated and Decrease in the Value Function for Various POMDP Tolerances. This chart plots the decrease in the POMDP value function from the optimal exact solution for several different POMDP tolerances. The numbers are the decrease from $val_9(0)$, the optimal expected payoff for a live target at the start of a 9-period horizon. We also plot the number of vectors in the policy for each tolerance. This demonstrates the tradeoff between accuracy and POMDP computation as measured by the number of vectors generated.

We stress in this section that the POMDPs should not be solved exactly, but we have not offered any advice on how to set the POMDP tolerance other than by experimentation. Running the decomposition with a tolerance of 10^{-2} yields substantially the same lower bound objective function value as running it with 10^{-9}, but at the cost of generating a much looser upper bound.

We offer a control in Section IV.C.3 that addresses this issue. But before we develop that scheme, we present a very simple acceleration method.

2. Solving POMDP Subproblems to Improvement

In Section III.C, we mention that many commercial LP packages use very fast pricing schemes in early iterations because it is relatively easy to improve the objective function. We can exploit the same idea in the decomposition.

After computing the POMDP recursion for any stage $n < T$, we can immediately check to see if it can improve the solution. This is because we allow pause actions in any period and assume a target cannot change state unless attacked. In this chapter, we assume all targets start in belief state 0 (live). Therefore, if $val_n'^g(0) - td_g > 0$ for any stage n, we can set the policy to pause for all belief states for stages n+1, ... T, stop the POMDP, and provide the column to the master LP.

Unfortunately, using this procedure makes our upper bound invalid. When we stop the POMDP prior to the final period T, we have not searched all the policies in S_g, and Theorem 2.1 does not hold. The upper bound (4.23) becomes an upper bound to a problem that only searches some subset of S.

Nevertheless, we can use this heuristic upper bound to decide when to start solving the POMDPs to optimality. In our implementation, we specify a number we call the *crossover setting*, which is analogous to the gap tolerance Ξ used to terminate the decomposition. While solving the POMDPs to improvement instead of optimality, we still compute the gap using (4.20). When this heuristic gap reaches the crossover setting, however, we solve all POMDPs to optimality for the rest of the iterations.

This very simple modification improves runtimes substantially. Using a crossover setting of 0.05 for the sensor-shooter example with a POMDP tolerance of 10^{-4} reduces the total number of iterations by 16% (82 versus 98) and reduces total runtime by 25% (882 seconds versus 1,179 seconds). The decomposition did not reach the crossover gap until the 19th iteration, so it was able to solve the POMDPs economically for a significant number of iterations.

We cannot offer any analytical scheme for setting the crossover value. It should be larger than the overall decomposition tolerance, but not so large that the decomposition solves the subproblems to optimality prematurely. We fix the crossover setting to 0.05 (5%) for the rest of this dissertation; this is substantially larger than the decomposition gap tolerance of 0.001 (0.1%).

3. Epsilon Control

Table 4.1 and Figure 4.4 raise the question of whether it is worthwhile to use the upper bound as a termination criterion. In all our testing, the progress of the decomposition has matched that shown in Figure 4.4; the majority of the runtime is spent gaining the last few percent of improvement. If we can be assured of slow tail convergence, then an appropriate stopping criterion might be to stop when the lower bound improves by less than some tolerance.

Nevertheless, the upper bound is essentially free, as we can compute it immediately from the POMDPs. Also, if we do not try to reduce the upper bound, we can never be sure how good the lower bound solution actually is. The question is whether or not we can reduce the upper bound without paying an unreasonable computational price.

The results of using the crossover setting in the last section are apparently contradictory. Table 4.1 shows we obtain essentially the same lower bound objective function value in 15% of the runtime by increasing the POMDP tolerance. Yet, reducing the POMDP tolerance from 10^{-9} to 10^{-4} does not change the number of iterations. On the other hand, using the crossover setting in Section IV.C.2 results in a substantial decrease in the number of iterations. This does not seem to make sense, given that the policies produced by the POMDP subproblems in the initial iterations have worse reduced costs than they would have if the subproblems were solved to optimality.

This phenomenon has a long history, and is described as a parable by Dantzig (1963, pp. 455-465). In any "price-directive" decomposition, the master problem can be viewed as a central planning agency that is trying to control resource expenditure among its manufacturing plants by setting prices. The plants, which are analogous to the POMDP subproblems, offer proposals for resource use based on the prices set by the central authority. Dantzig describes decomposition as the sequence of prices and proposals between the central authority and the plants, which stops when the resource prices stabilize.

Consider what happens in Dantzig's example when the central authority puts a low price on a resource. That resource becomes attractive to the plants, and they rush in with proposals demanding much of that resource. The central authority panics at the possibility of wanton overconsumption, and raises the resource price drastically.

These oscillations in prices are typical in decomposition schemes, and they can seriously slow down convergence. This behavior is analogous to that of the popular nonlinear optimization technique known as "steepest descent," which exhibits similar oscillations and has poor convergence properties (e.g., Bazarra, Shetty, and Sherali, 1993, pp. 303-304). Brown, Graves, and Honczarenko (1987, pp. 1475-1477) comment on this issue with respect to decomposition in their paper on the optimization of a production and distribution system, and suggest moderating the subproblems' tendency to compute extreme solutions when all that is needed are improving solutions.

The crossover setting is one such moderation scheme, and reduced POMDP tolerances can be viewed as another. The latter method comes at some cost, as the reduced tolerance also leads to a looser upper bound and more uncertainty about the worth of the lower bound solution in any iteration.

Now, there is no reason that the POMDP tolerance has to remain the same throughout the decomposition. Early on, we can afford to keep the tolerances loose while we are still generating improving columns. Later, when we are not improving the lower bound much but trying to reduce the upper bound, we can tighten the tolerances.

Furthermore, we are solving POMDPs for each object class, so there is no reason that the tolerance has to be the same for each class. In the sensor-shooter example, the total available value in each target class ranges from 1 to 7,920, so we should be more interested in targets of the latter class than the former.

We call any such scheme that systematically adjusts the POMDP tolerance during the decomposition an *epsilon control* method. We suggest one control scheme (of many we tried) that works well for the sensor-shooter example, makes sense in general, and requires no additional parameters other than the problem data.

We determine the POMDP tolerances ε_k^g for iteration k as follows. Since the decomposition starts with the master LP, we use the tolerances with index k-1 to solve the POMDPs in the kth iteration. Following (4.32), we define the POMDP error as

$$errptot_k \equiv \sum_{g \in G} \left(TGT_g \sum_{t=1}^{T} eps_t^g \left(\varepsilon_{k-1}^g \right) \right). \tag{4.34}$$

The heuristic upper bound, which is not adjusted for POMDP error, is

$$\underline{u}\left(T^k; \lambda^k\right) \equiv \begin{Bmatrix} \sum_{i \in I} \sum_{t} sd_{it}\, SORT_{it} + \sum_{w \in W} wd_w\, WPN_w + \\ \sum_{o \in O} \sum_{t} ld_{ot}\, LOOK_{ot} + \sum_{i \in I} ad_i\, MAXATT_i + \\ \sum_{g \in G} TGT_g\, \underline{val}_T^g\left[\pi(1), \lambda^k\right] \end{Bmatrix}. \tag{4.35}$$

The global upper bound, which is a true upper bound, is

$$\overline{u}\left(T^k;\lambda^k\right)=\min_{m=1\ldots k}\left\{\underline{u}\left(T^m;\lambda^m\right)+errptot_k\right\}. \tag{4.36}$$

Our aim in this heuristic is to divide the decomposition gap into the sum of two quantities, the "dual gap" and the "POMDP gap." We estimate the dual gap by the quantity $\underline{u}\left(T^k;\lambda^k\right)-v\left(T^k\right)$, and the POMDP gap by the quantity $\overline{u}\left(T^k;\lambda^k\right)-\underline{u}\left(T^k;\lambda^k\right)$. Then, we use the ratio of these quantities to decide whether to shrink the POMDP tolerances. If the dual gap is bigger than the POMDP gap, then it is more efficient to allow the LP to continue to adjust dual prices; otherwise, we need to reduce the POMDP tolerances.

Note that the POMDP gap can be negative, so we form the following quantity:

$$Pgap_k=\left\{\begin{matrix}\underline{u}\left(T^k;\lambda^k\right)-v\left(T^k\right), & \overline{u}\left(T^k;\lambda^k\right)-\underline{u}\left(T^k;\lambda^k\right)\le 0\\ \overline{u}\left(T^k;\lambda^k\right)-\underline{u}\left(T^k;\lambda^k\right), & \text{otherwise}\end{matrix}\right\}. \tag{4.37}$$

With this convention, our epsilon control scheme for the sensor-shooter problem is

$$\begin{aligned}\varepsilon_0^g&=\min\left\{\frac{V_g}{2},\frac{V_g}{TGT_g}\right\},\\ \varepsilon_k^g&=\min\left\{\varepsilon_{k-1}^g,\ \varepsilon_{k-1}^g\cdot\frac{\underline{u}\left(T^k;\lambda^k\right)-v\left(T^k\right)}{Pgap_k}\right\},\quad k=1,2,\ldots\end{aligned} \tag{4.38}$$

In addition to being simple, (4.38) makes intuitive sense. We initialize the tolerances to be larger if the target values are larger, but reduce them if there are many targets of that type. We make the tolerance at most half of the target's value to make sure the POMDPs simply do not generate "all-pause" policies. The updates adjust the tolerances by the ratio of the gap estimates, but do not allow the tolerance to increase. This scheme matches our philosophy of solving the POMDPs using a large tolerance; we only need be as accurate as the resource prices.

Table 4.2 reports the same information as Table 4.1 for a POMDP tolerance of 10^{-4} with and without crossover, the epsilon control scheme (4.38) with and without crossover, and a

POMDP tolerance of 10^{-2} with crossover. When we combine crossover with an epsilon control scheme, we do not start updating the POMDP tolerances until the decomposition reaches the crossover point and begins solving the subproblems to optimality. This is because we cannot legitimately compute (4.36) without a global upper bound.

Tolerance	lower	upper	POMDPs	LPs	Total
0.0001	61137.56	61185.16	1062	86	1179
0.0001 (cr)	61139.89	61197.09	809	50	882
0.01 (cr)	61145.57	61375.43	245	64	338
control	61146.42	61201.13	171	55	252
control (cr)	61143.75	61203.16	145	34	199

POMDP Tolerance	POMDP error	# vectors in max policy	policies generated	iterations
0.0001	1.73	516	4548	98
0.0001 (cr)	1.73	527	3482	82
0.01 (cr)	229.36	138	3695	>100
control	25.78	361	3308	90
control (cr)	29.76	328	2786	76

Table 4.2 Decomposition Solution Statistics for Various POMDP Error Tolerance Schemes. These two tables duplicate most of Figure 4.5, but compare using an epsilon control scheme and crossover to fixing the POMDP tolerances. The cases designated (cr) use a crossover setting of 5%, and the cases designated "control" use the scheme shown in (4.28) for computing POMDP tolerances. The control cases dominate the others in terms of runtime.

The epsilon control schemes dominate the other alternatives in terms of computing time. Combining epsilon control with crossover reduces the total runtime to 199 seconds and takes 76 iterations. The fixed schemes with tighter tolerances can provide the tight upper bound we want, but require four times as much solution time. The coarser tolerance settings give comparable lower bounds, but cannot reduce the upper bound beyond a certain level.

In the sensor-shooter example, the initial tolerances range from 0.011 to 19.33 with an average of 1.16. At the end of the decomposition using crossover, they were reduced by 99.53%, so they range from 0.000051 to 0.089, with an average of 0.0054.

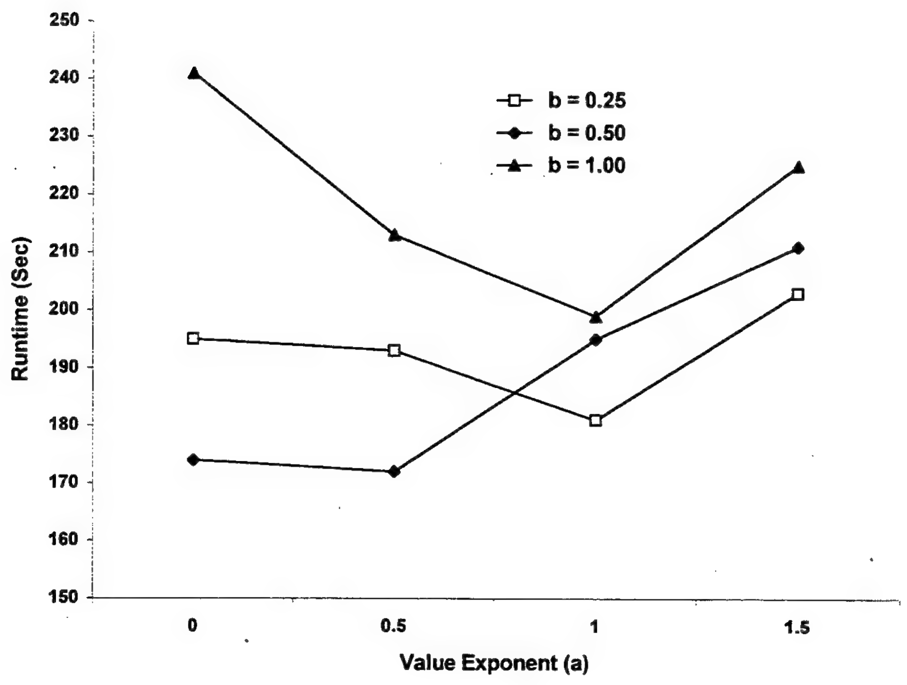

Figure 4.6: Runtimes for the Sensor-Shooter Example Using Various Exponent Values to Determine the Initial POMDP Tolerances. This chart shows the runtimes for various settings of the exponents *a* and *b* used to calculate $\varepsilon_0^g = (V_g)^a / (TGT_g)^b$. While it is possible to improve runtimes by varying these settings, there is no way to determine them beforehand. We use the settings *a* = *b* = 1 for the results in this section.

There are many alternatives to (4.38). We have tried various formulas and even attempted to solve for the tolerances using nonlinear optimization, but were not able to achieve results substantially different than those achieved using (4.38). Figure 4.6 shows the results of setting $\varepsilon_0^g = (V_g)^a / (TGT_g)^b$ for different combinations of the exponents a and b. While there are several combinations that provide better results than the $a = b = 1$ that we recommend, there is really no way to determine these settings beforehand, and they are certainly data-dependent. If the model were being run for many trials with similar data, doing an analysis similar to the one shown in Figure 4.6 might be useful. For a few runs, it is probably not worth the effort.

4. Limiting POMDP Actions

Recall from the proof of Theorem 3.1 that the maximum number of α-vectors we can generate in any stage n of a POMDP is given by $|A||V(n-1)|^{|\Theta|}$, where V(n-1) is the set of α-vectors used to determine the value function in stage n-1. If we can keep $|A|$ and particularly $|\Theta|$ small, we can generally solve the POMDPs more quickly.

Since we are solving sequences of POMDPs, we should learn something from the previous subproblem solutions about which actions were used for which object classes. In the sensor-shooter problem, only about 400 tactics out of 5,203 are ever used in any policy, and after the initial iterations, the sets of actions used in the policies for each target class do not change much. We already eliminate some actions for each target based on their resource cost using Theorem 4.2, Theorem 4.3, and Corollary 4.5; we now propose a stronger filter.

Our *limiting control* works as follows. We keep track of which actions are used for each target class each time we generate a policy. Define $A_k^g \subseteq A_g$ as the set of actions used in any policy generated in iterations 1, 2, ... k. We specify a number called the *update interval*, denoted *ui*, which is the number of decomposition iterations that we restrict the actions to those in A_k^g. Every *ui* + 1 iterations, we solve the POMDPs using all possible actions. So, if the update interval were 10, we would use all POMDP actions in the first iteration and construct A_1^g for all target types g. We would use these sets of actions for 10 iterations, then solve the POMDPs using all actions in iteration 11. We would then update the action sets to A_{11}^g, use them for the next 10 iterations, and so on.

As with solving the POMDPs to improvement, limiting the actions available to the POMDPs makes the upper bound computation a heuristic bound. Nevertheless, the lower bound improves as long as at least one subproblem produces an improving policy.

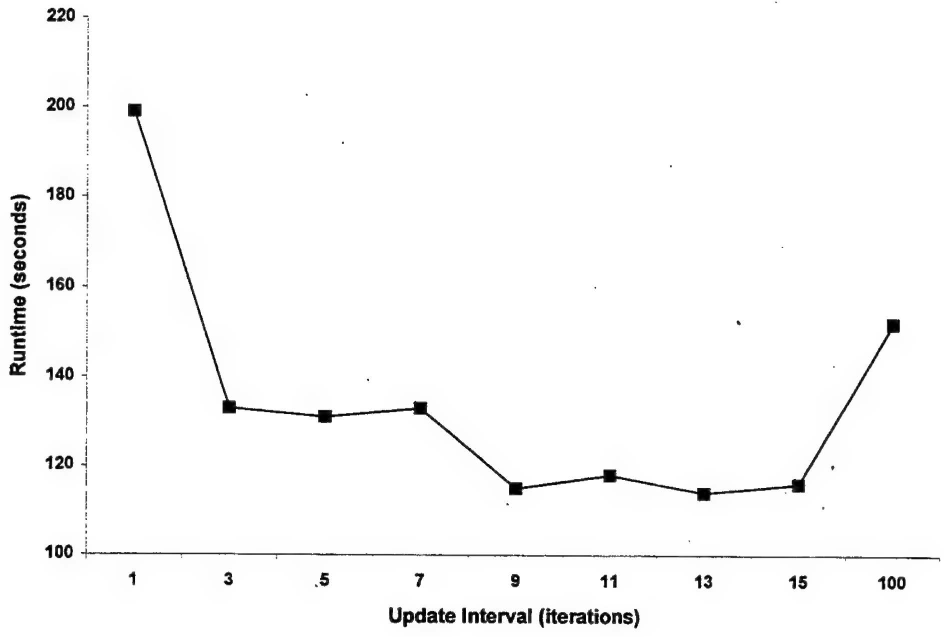

Figure 4.7: Runtime as a Function of the Update Interval for the Sensor-Shooter Example. This chart shows total runtimes as a function of the interval between updates of the actions available to each target. In other iterations, the heuristic restricts the actions available to each POMDP to those used in previous policies, and only considers all actions in stages determined by the update interval. As the decomposition gets close to meeting the required gap, the heuristic considers all actions in every iteration. Using this heuristic significantly improves runtimes, even with intervals as large as 100.

We make two adjustments in the implementation of this control. First, if we have set a crossover gap, we do not limit the actions until the decomposition starts solving the subproblems to optimality. If the crossover gap is met in iteration k, we solve the subproblems with restricted sets of actions in iteration $k + 1$, and do not update again until iteration $k + 1 + ui$. Second, we always calculate the heuristic upper bound using (4.23). If we restrict the actions and the heuristic decomposition gap drops below $\Xi/2$ (half of the gap tolerance), we solve the POMDPs with all actions for the remainder of the algorithm. This prevents situations where we could reach the specified gap tolerance in one more iteration, but would otherwise have to wait *ui* more iterations to compute a legitimate upper bound.

Figure 4.7 shows the effects of adjusting the update interval for the sensor-shooter example using epsilon control (4.38) and a crossover setting of 5%. Moving from an interval of 1 (no restrictions on actions in any iteration) to an interval of 5 reduces runtime from 199 seconds to 131 seconds, a 34% reduction. The minimum runtime occurs with an interval of 13, after which the runtimes begin to increase. Setting the update interval to 100 (which means we do not update until the heuristic gap is half of the decomposition gap tolerance), still improves the runtime by 24% (152 seconds versus 199).

We do not know how to determine a good update interval beforehand, but it is much better to set it at a high value than a low one. We would also expect dramatic reductions in runtimes in models with more complicated subproblems. In many POMDPs the possible observations are related to the choice of action, so throwing out actions may also implicitly reduce $|\Theta|$ and simplify solving the POMDP.

5. Contributions of the POMDP Acceleration Controls

The combined effects of the 3 controls – solving to improvement, epsilon control, and restricting actions – reduce the total runtime in the sensor-shooter example by over 90% from the initial attempt with exact POMDP solutions. But, we must measure the effects of each of the controls while accounting for the effects of the others to determine the relationships among them.

To assess the individual contributions of the 3 controls, we designed a small experiment to test each of them at two levels. The epsilon control settings are "on" as in (4.22) versus "off," which we specify as fixing the POMDP tolerances at 0.001. For crossover, we used settings of 0.05 and no crossover, and for the update interval, we used settings of 1 iteration (no control) and 11 iterations.

POMDP Tolerance	Crossover	Update Interval	Runtime
epsilon control	5%	11	118
epsilon control	5%	1	199
epsilon control	off	11	122
epsilon control	off	1	251
0.001	5%	11	243
0.001	5%	1	511
0.001	off	11	282
0.001	off	1	771

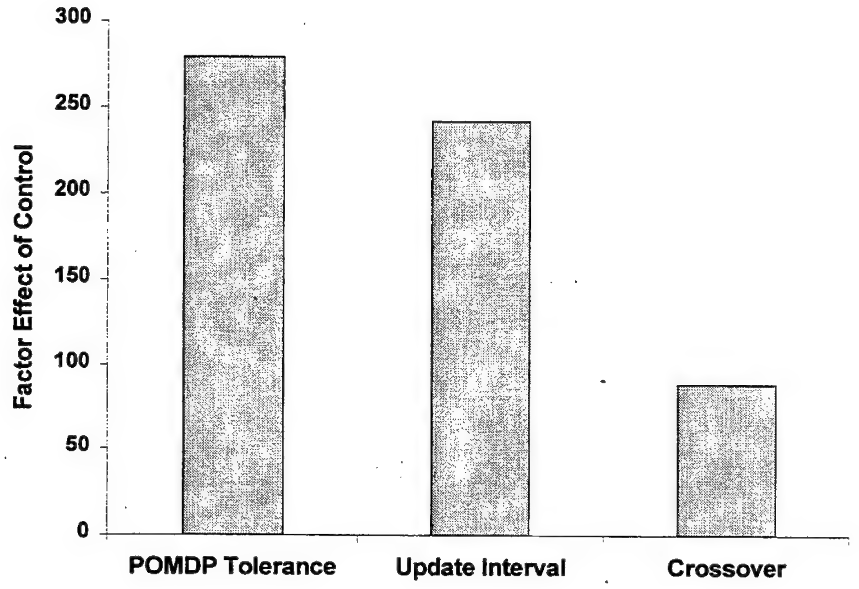

Figure 4.8: Factor Effects of Controls. The table above shows runtimes for the sensor-shooter example with all combinations of each control set at two levels. The "factor effect" is the mean change in the runtime over the two settings for each control. Epsilon control and long update intervals give large reductions; using crossover results in smaller, but still significant, improvements.

In Figure 4.8, we show the runtimes from all $2^3 = 8$ possible combinations for the sensor-shooter example. This "factorial design" is a useful way to measure the effect of each control while adjusting for the settings of the others. Also, this approach is a good way to determine relationships between the controls.

We also plot the "factor effects," which are the absolute differences in the marginal mean runtimes for each control at each level. For example, the average runtime for the cases using epsilon control is 172.5 seconds; the average for the cases with the POMDP tolerance fixed at 0.001 (that is, with no epsilon control) is 471.75. The absolute difference of these two means, 279.25, is plotted as the factor effect. This measures the change in the mean runtime for all cases (which is 312.125) over the two tolerance settings.

The table in Figure 4.8 shows there are no conflicting "interactions" between the controls. While the effects of the controls are related, there are no pairwise combinations that decrease the performance of the algorithm over using a single control on its own.

While the effects we have shown in this section only apply to one instance of one problem, we see no reason to believe they will not apply to any decomposition of the class we have developed. The characteristics of the POMDP make finding an exact solution difficult, so any legitimate approximations – solving only to improvement, reducing the tolerances only when necessary, limiting the actions – can only help the overall decomposition. In addition, these controls moderate the negotiation between the master LP and the subproblems over the resource costs, preventing time-consuming oscillations. We speculate that decompositions with more complicated POMDP subproblems could be accelerated even more than the 95% improvement we have demonstrated in this example.

We close this section by reminding the reader that the algorithmic modifications in this section all lead to optimality. We continue to generate valid upper and lower bounds throughout the decomposition, and terminate with the same tolerances.

D. ACCELERATING THE MASTER PROBLEM

We also made two attempts to reduce time spent solving the master LPs. Unfortunately, one method had little effect, and the other actually increased solution times. We present both; we feel the first method may be useful for decompositions with more complex master problems, and we offer our experience with the second as a test of some of the claims in the optimization literature.

1. Column Control

In the experiments in Section IV.C, we retained all generated policies (columns) in the master LP. Nevertheless, the master LP only contains 281 constraints, so we know from (2.3) that an optimal basic solution exists that uses at most 281 policies.

Furthermore, some policies will never price favorably once the dual resource costs begin to stabilize, and keeping them in the LP merely creates excess work. The risk of removing such a column is that one of the subproblems may generate it again. If a column has not priced favorably for many iterations, however, we can probably remove it from the master problem with no risk.

The simplest way to control the columns in the master LP would be to set some maximum number of columns *C*. Then, we would store all columns generated in the decomposition and recompute their reduced costs after every master LP solution. After generating *N* potentially-improving columns from the subproblems, we would pick the best *C-N* of the stored columns based on current reduced costs to use in master LP in the next iteration.

Our approach is similar, but we again opt for moderation. Instead of using current reduced costs, we maintain a separate *exponentially-smoothed* reduced cost for every LP column. Let $rc(s,k)$ be the reduced cost for policy s after solving the master LP in iteration k, and let $src(s,k)$ be the exponentially-smoothed reduced cost. Denote the first iteration the policy is used in the master LP as k_s, and specify some smoothing parameter $0 < m < 1$. Then, the smoothed reduced costs are:

$$\begin{aligned} &src(s,k_s) = rc(s,k_s)\ ; \\ &src(s,k) = (1-m)\cdot src(s,k-1) + m\cdot rc(s,k), \quad k > k_s . \end{aligned} \tag{4.39}$$

The smoothing parameter *m* controls how fast the smoothed price reacts to new pricing information. Setting $m = 1$ gives the simple scheme we first proposed, while setting $m = 0$ means the smoothed price never reacts.

smoothing parameter (*m*)	maximum columns (*C*)	Runtimes (seconds)		
		Master LPs	POMDPs	Total
0.9	2500	27	68	123
0.9	3500	30	57	110
0.9	4500	34	64	121
0.5	2500	26	64	115
0.5	3500	29	62	114
0.5	4500	34	63	119
0.1	2500	26	65	114
0.1	3500	31	65	120
0.1	4500	34	64	120

Table 4.3: Runtimes for Various Settings of the Smoothing Parameter and the Maximum Number of Master LP Columns. The choosing of smoothing parameter has almost no effect on any runtimes in the sensor-shooter example, and restricting the number of master LP columns has little effect. The greatest difference in total runtime among the 9 cases is only 13 seconds. These runs used a crossover setting of 0.05 and an update interval of 11.

Table 4.3 gives the results of 9 trials of the sensor-shooter example, using a crossover setting of 0.05 and an update interval of 11. We constrain the total number of columns to be either 2,500, 3,500, or 4,500, because the heuristic generates 2,214 columns initially, and the decomposition, if left unconstrained, tends to terminate with about 5,000 total columns for the sample problem.

The gains are very small. There is only a 13-second difference in total runtime from the best to the worst case, and an 8-second difference in the total LP solution time. Furthermore, the smoothing parameter does not seem to have any effect; all of the change is due the maximum column setting.

Although we cannot report any exciting empirical results, we would recommend considering this method for decompositions of this type. The number of potential columns is generally so enormous that controlling the number generated is a good idea.

2. Using Interior-Point Methods for the Master Problem

Since their introduction by Karmarkar (1984), interior-point methods for linear programming have generated an enormous amount of research. Nearly all commercial LP packages implement some version of an interior-point method, and these algorithms have become mainstream tools.

One of the characteristics of interior-point methods that has attracted researchers is their tendency, in the case of multiple optima, to produce a solution that is in the center of the optimal "face" of the polyhedral set of constraints. This is in contrast to the simplex method, which stops at some vertex of the optimal face and reports a basic solution.

Several presenters at recent conferences (e.g., Greenberg 1998, Holder 1998) have claimed that such interior-point solutions provide better dual information, because the dual representation of a point in the center of the optimal face better represents the entire face. This idea was also advanced in a recent article by Barnhart et al. (1998, p. 325), who temper their advice by commenting that no one seems to have tested this proposition.

Since our decomposition relies on dual resource costs and has a potentially huge number of columns, it seems reasonable that our solutions would contain multiple optima. Our situation seemed to match the cases described by the researchers above, so we conducted an experiment to determine the effects of using an interior-point method instead of simplex for the master problem.

Unfortunately, interior point methods have a considerable drawback: they cannot restart from a previous solution. Adding new columns in the simplex algorithm is a very efficient procedure, and simplex can compute the new optimum very quickly. In contrast, interior-point methods work by computing a sequence of projections, which means they must essentially start over if new columns are added. The result from our testing was a 770% increase in master problem solution times, from 27 seconds to 236 seconds, and a decrease of 1 iteration in the decomposition.

It may be that problems exist that can take advantage of the "central" solutions produced by interior-point algorithms. Nonetheless, their inability to quickly reoptimize from a previous solution makes them computationally suspect in any column-generation scheme, and it seems that an interior-point method would have to generate overwhelmingly-better dual information to overcome this drawback.

E. SUMMARY

In this chapter, we demonstrate that the convergence and finiteness of the decomposition proved in Chapter II is not just theoretical. The sensor-shooter example is a real problem, and simply cannot be solved as either a monolithic LP or a POMDP.

Most of the time in the decomposition is consumed with solving the POMDPs. Nevertheless, we show in this chapter that by using an approximate POMDP algorithm and keeping the tolerances as loose as possible, we can speed up the decomposition by an order of magnitude and still produce a solution that meets the specified optimality gap. We gain the reductions through 3 controls, none of which is mathematically demanding or difficult to implement.

We are less successful at reducing the solution times of the master LPs. After improving the performance of the subproblems, the master LPs consume close to 30% of the total runtime, so their overhead is significant. Nonetheless, there appears to be little room for improvement, particularly in the sensor-shooter example.

V. APPLYING THE DECOMPOSITION IN A STOCHASTIC ENVIRONMENT

In Chapter I, we began our definition of the original problem by writing the following mathematical program:

$$\max_{x} \sum_{s\in S, j\in J} R_{sj} x_{sj} \tag{5.1}$$

$$\begin{aligned} &\sum_{s\in S, j\in J} Y_{sji}\, x_{sj} \le b_i \quad \forall\, i \in I \\ &\sum_{s\in S} x_{sj} = N_j \quad \forall\, j \in J \\ &x_{sj} \text{ integer } \quad \forall\, s \in S, j \in J. \end{aligned} \tag{5.2}$$

This problem is not well-defined. We do not know how to maximize by choosing x prior to knowing the values of the random parameters R_{sj} and Y_{sji}.

By replacing R_{sj} and Y_{sji} with their expectations, we create a legitimate optimization problem and solve it using the decomposition. Furthermore, this optimization meets our requirements of satisfying the constraints on the average and maximizing the total expected payoff.

Nevertheless, the sensor-shooter problem that motivated this research may have more strict constraints. In a large-scale campaign, we only have so many sorties, so many weapons, and so many sensor looks. Many of these resources are highly specialized and very scarce, and most of them cannot be augmented in a contingency at any price. An example of this is the GBU-28 laser-guided bomb, which was developed in a crash program shortly prior to the start of DESERT STORM and was built specifically to attack deeply-buried bunkers. Due to its method of construction (which involved packing a modified howitzer barrel with explosives by hand), only 4 of these bombs were available, and the first two were used for flight testing. Constraining the average consumption for such a weapon may yield infeasible solutions, because we simply cannot obtain any more of them.

We call such resources *rigid*, and in this chapter we concentrate on solving a general problem that uses resources of this type. Specifically, this *rigid problem* has two sets of constraints: those associated with rigid resources $\left(I^{rigid}\right)$; and those associated with "soft" resources $\left(I^{soft}\right)$. The problem we outlined in Chapter I contains nothing but soft constraints, i.e., those we could constrain in expectation.

In Section V.A, we describe the rigid problem and develop analytical bounds on its optimal objective function value using the decomposition. We also develop a simulation to estimate the distributions of outcomes for the rigid problem, and show that the structure of the decomposition makes this simulation computationally feasible.

In Section V.B, we apply the simulation to a targeting problem analyzed by Aviv and Kress (1997) and compare our results to their known analytical solution. In Section V.C, we apply the simulation approach to the sensor-shooter example and report empirical results.

A. APPLYING THE DECOMPOSITION TO THE RIGID PROBLEM

1. Rigid Problem Formulation

In Chapter II, we defined the set of policies $S \equiv \bigcup_{j \in J} S_j$, and only considered policies that controlled individual objects. This allowed us to limit the object states in the POMDP subproblems to a single object.

Unfortunately, the extension to a rigid problem means that assigning resources to one object can affect the actions we take for the others. This means we must consider all joint policies that map from the belief states for all objects to the actions. Even worse, we must add the amounts of each rigid resource available to the state space, because the amount of each rigid resource remaining is assumed to be observable and affects the actions we take for each object. We define the set of policies that map from the belief states of the objects and the amounts of the rigid resources remaining to the set of actions as S^*.

Additionally, let K_j be the set of all objects in class j. With this notation, we denote the rigid problem as $SPR(S^*)$, whose objective is to find a single optimal joint policy:

$$\begin{aligned} SPR(S^*):\quad & \max_{s\in S^*} \sum_{j\in J}\sum_{k\in K_j} E\left(R_{sjk}\right) \\ st\quad & \sum_{j\in J}\sum_{k\in K_j} Y_{sjki} \le b_i \quad \forall\, i \in I^{rigid} \text{ with probability 1} \\ & \sum_{j\in J}\sum_{k\in K_j} E\left(Y_{sjki}\right) \le b_i \quad \forall\, i \in I^{soft} \\ & I = I^{rigid} + I^{soft}, I^{rigid} \cap I^{soft} = \varnothing. \end{aligned} \tag{5.3}$$

2. Objective Function Bounds

We cannot solve a POMDP using S^*. In the sensor-shooter problem, the state space for the 6,316 targets would contain 2^{6316} possibilities, and adding states for the amounts remaining of any resource we make rigid only adds to the intractability. This is exactly the problem confronted by Meleau et al. (1998) in their analysis of a similar aircraft-weapon-target allocation problem with perfect observability. The state space is too large to be manageable for any but the smallest of problems.

Nonetheless, we can determine an upper bound on the objective function value of $SPR(S^*)$. In the following, the function $v[\cdot]$ denotes the optimal objective function value of the problem enclosed in the brackets. We begin with the following lemma:

Lemma 5.1: Let *SPRL*(S^*;λ) be defined as follows:

$$SPRL\left(S^*:\lambda\right): \quad \max_{s\in S^*}\left\{\sum_{j\in J}\sum_{k\in K_j}E\left(R_{sjk}\right)+\sum_{i\in I}\lambda_i\left[b_i-\sum_{j\in J}\sum_{k\in K_j}E\left(Y_{sjki}\right)\right]\right\} \qquad \textbf{(5.4)}$$

$$\lambda \equiv \left(\lambda_i : \lambda_i \geq 0 \quad \forall\, i \in I\right).$$

Then, v[*SPRL*(S^*;λ)] ≥ *v*[*SPR*(*S)] for all λ ≥ 0.**

Proof: The proof follows the same arguments as Theorem 2.1. Replace all the rigid constraints in *SPR*(*S**) with their expectations. The resulting optimization is a relaxation of *SPR*(*S**), so its optimal objective value must be larger than *v*[*SPR*(*S**)]. Now, add the term $\sum_{i\in I}\lambda_i\left[b_i-\sum_{j\in J}\sum_{k}E\left(Y_{sjki}\right)\right]$ to the objective function of this new optimization. Since the constraints must be satisfied and λ is nonnegative, this term cannot increase the objective function value. Finally, remove the constraints. We cannot decrease the objective function value by removing constraints, and the resulting optimization is *SPRL*(S^*;λ). Therefore, *v*[*SPRL*(S^*;λ)] ≥ *v*[*SPR*(*S**)] for all λ ≥ 0. ■

Lemma 5.2: Define *LPU2*(*S*;λ) as in Section II.C. Then *v*[*LPU2*(*S*;λ)] ≥ *v*[*SPR*(*S)] for all λ ≥ 0.**

Proof: We return to the notation of Section III.A.1. for this proof. Let $\underline{\Pi}=\Pi_{1,1}\times\Pi_{1,2}\times\ldots\times\Pi_{1,|K_1|}\times\ldots\times\Pi_{|J|,|K_{|J|}|}$ be the belief space of the joint POMDP with components Π_{jk}, and denote the belief state as $\underline{\pi}=\left(\pi_{jk}\right)$. Also, denote the set of joint actions as $\underline{A}=A_{1,1}\times A_{1,2}\times\ldots\times A_{1,|K_1|}\times\ldots\times A_{|J|,|K_{|J|}|}$. Define $Tr\left(\underline{\pi}|\underline{\theta},\underline{a}\right)$ be the joint transformation if we take the joint action $\underline{a}=\left(a_{jk}\right)$ and observe the joint observation $\underline{\theta}=\left(\theta_{jk}\right)$, and let the terminal rewards for each object be the vectors $Re^{jk} = \left(r_e^{jk}\right)$ and the action costs be given by the vectors $Ra^{jk} = \left(r_a^{jk}\right)$. Using this notation, we define the DP recursion for the value function for this joint POMDP as

$$valj_0(\underline{\pi}) = \sum_{j \in J} \sum_{k \in K_j} \pi_{jk} \cdot Re^{jk},$$
$$valj_n(\underline{\pi}) = \max_{\underline{a} \in \underline{A}} \left\{ \sum_{j \in J} \sum_{k \in K_j} -\pi_{jk} \cdot Ra^{jk} + E\left[valj_{n-1}\left(Tr[\underline{\pi} | \underline{\theta}, \underline{a}]\right) \right] \right\}, \tag{5.5}$$
$$n = 1, 2, \ldots T.$$

Compare this to the individual object POMDPs from Section III.A.1:

$$val_0^{jk}\left(\pi_{jk}\right) = \pi_{jk} \cdot Re^{jk},$$
$$val_n^{jk}\left(\pi_{jk}\right) = \max_{a \in A_{jk}} \left\{ -\pi_{jk} \cdot Ra^{jk} + E\left[val_{n-1}^{jk}\left(Tr\left[\pi_{jk} \left| a, \theta\right.\right]\right) \right] \right\}, \tag{5.6}$$
$$n = 1, 2, \ldots T.$$

We assume that $Tr(\underline{\pi} | \underline{\theta}, \underline{a})_{jk} = Tr\left(\pi_{jk} | \theta_{jk}, a_{jk}\right)$; that is, the belief states are updated identically for the same action and observation regardless of whether we are executing joint or independent policies.

We first show by induction that

$$valj_n(\underline{\pi}) \le \sum_{j \in J} \sum_{k \in K_j} val_n^{jk}\left(\pi_{jk}\right), \quad n = 0, 1, \ldots T. \tag{5.7}$$

The inequality in (5.7) is true for $n = 0$ by the definitions in (5.5) and (5.6). Rewrite (5.5) as follows:

$$valj_n(\underline{\pi}) = \max_{\underline{a} \in \underline{A}} \left\{ E\left[\sum_{j \in J} \sum_{k \in K_j} -Ra^{jk} + valj_{n-1}\left(Tr[\underline{\pi} | \underline{\theta}, \underline{a}]\right) \right] \right\}. \tag{5.8}$$

Applying the induction hypothesis and the independence assumptions gives

$$valj_n(\underline{\pi}) \le \max_{\underline{a}\in\underline{A}}\left\{E\left[\sum_{j\in J}\sum_{k\in K_j}\left[-Ra^{jk} + val_{n-1}^{jk}\left(Tr\left[\pi_{jk}|\theta_{jk}, a_{jk}\right]\right)\right]\right]\right\}. \tag{5.9}$$

Since $\underline{A}$ is a cross-product, the expectation and maximization operations commute:

$$valj_n(\underline{\pi}) \le E\left[\sum_{j\in J}\sum_{k\in K_j}\max_{\underline{a}\in\underline{A}}\left\{-Ra^{jk} + val_{n-1}^{jk}\left(Tr\left[\pi_{jk}|\theta_{jk}, a_{jk}\right]\right)\right\}\right]. \tag{5.10}$$

The terms for the *jk*th object only depend on a_{jk}, so,

$$\begin{aligned} valj_n(\underline{\pi}) &\le E\left[\sum_{j\in J}\sum_{k\in K_j}\max_{a\in A_{jk}}\left\{-Ra^{jk} + val_{n-1}^{jk}\left(Tr\left[\pi_{jk}|\theta_{jk}, a_{jk}\right]\right)\right\}\right] \\ &\le \sum_{j\in J}\sum_{k\in K_j}\max_{a\in A_{jk}}\left\{-\pi_{jk}\cdot Ra^{jk} + E\left[val_{n-1}^{jk}\left(Tr\left[\pi_{jk}|\theta_{jk}, a_{jk}\right]\right)\right]\right\} \\ &\le \sum_{j\in J}\sum_{k\in K_j} val_n^{jk}\left(\pi_{jk}\right). \end{aligned} \tag{5.11}$$

Therefore, the relationship is true for all *n*, and we can get as much value by using individual policies as we can using any joint policy.

Now, rewrite the above result using Theorem 2.4:

$$\begin{aligned} &\max_{s\in S^*}\sum_{j\in J}\sum_{k\in K_j}\left[E\left(R_{sjk}\right) - \sum_{i\in I}\lambda_i E\left(Y_{sjki}\right)\right] + \sum_{i\in I}\lambda_i b_i \le \\ &\sum_{j\in J}\sum_{k\in K_j}\max_{s\in S_j}\left\{E\left(R_{sjk}\right) - \sum_{i\in I}\lambda_i E\left(Y_{sjki}\right)\right\} + \sum_{i\in I}\lambda_i b_i. \end{aligned} \tag{5.12}$$

Since each object in class *j* is assumed to be identical, the maximizing policy in each set S_j maximizes the expected payoff of all objects in the class:

$$\sum_{j\in J}\sum_{k\in K_j}\max_{s\in S_j}\left\{E\left(R_{sjk}\right)-\sum_{i\in I}\lambda_i E\left(Y_{sjki}\right)\right\}+\sum_{i\in I}\lambda_i b_i = \sum_{j\in J}\left[N_j\cdot\max_{s\in S_j}\left\{E\left(R_{sjk}\right)-\sum_{i\in I}\lambda_i E\left(Y_{sjki}\right)\right\}\right]+\sum_{i\in I}\lambda_i b_i. \tag{5.13}$$

The latter expression is equivalent to *LPU2(S)*:

$$\begin{aligned} \text{LPU2}(S;\lambda):\quad & \max_x \sum_{j\in J}\sum_{s\in S_j} E\left(R_{sj}\right)x_{sj}+\sum_{i\in I}\lambda_i\left(b_i-\sum_{j\in J}\sum_{s\in S_j}E\left(Y_{sji}\right)x_{sj}\right) \\ & \text{st}\quad \sum_{s\in S}x_{sj}=N_j \\ & x_{sj}\ge 0 \quad \forall\, j\in J, s\in S_j. \end{aligned} \tag{5.14}$$

Therefore, $v[LPU2(S;\lambda)] = v[SPRL(S^*;\lambda)]$, and $v[LPU2(S;\lambda)] \ge v[SPR(S^*)]$ for all $\lambda \ge 0$ by Lemma 5.1. ■

Not surprisingly, the final theorem shows that we know how to solve the optimization that determines the bound:

Theorem 5.3: Let *LP2(S)* be as defined in Section II.C. Then, $v[LP2(S)] \ge v[SPR(S^*)]$.

Proof: By Lemma 2.2, there exists a $\lambda \ge 0$ such that $v[LP2(S)] = v[LPU2(S;\lambda)]$. The result follows by Lemma 5.2. ■

This result is not a particularly earthshaking revelation; we should be able to earn better rewards if we are allowed to violate constraints occasionally and control the objects independently. Fortunately, we have spent most of this dissertation determining how to solve *LP2(S)*, so the upper bound is readily available.

We can also obtain an analytic lower bound by constructing a particular subset of the policies in S and showing we can compute a feasible solution to $SPR(S^*)$ using those policies. This lower bound is not as useful as the upper bound, because we must remove the one thing we want to measure – the information resources – to compute the bound. Nonetheless, this bound does give us an analytic result that is easy to compute, and it allows us to quickly estimate the value of the information resources.

The key to our lower bound is constructing a set of "deterministic-consumption policies" $S^D(I) \subseteq S$:

Definition 5.1: A deterministic-consumption policy $s \in S^D(I)$ is an admissible policy for a POMDP which, when followed for any initial belief state, uses deterministic amounts of each resource $i \in I$.

Suppose we allow the attrition constraint to be soft in the sensor-shooter example, and make the other constraints rigid. Then the "single-shot" policies generated by the heuristic described in Section IV.A.7 are members of $S^D(I)$, as they consume a fixed number of sorties and weapons every time they are used. On the other hand, the "shoot-look-shoot" policies generated by the heuristic are not members of $S^D(I)$, because the second attack is scheduled based on the outcome of the random sensor look.

The question is whether or not we can induce the decomposition algorithm to generate only deterministic policies. There are two sources of randomness in a policy: the first is the action chosen in any stage; and the second is the consumption of resources. To produce a deterministic policy, we must address both sources of randomness.

For the actions, note that if the POMDP only uses actions that result in a single observation, no branching ever occurs in the associated decision tree, and the resulting policy for any starting belief state is just a sequence of scripted actions across the time horizon. In other words, each starting belief state only has a single possible sample path of actions.

We can transform any POMDP to a single-observation POMDP by "blinding" all possible actions and giving them a single null observation that occurs with probability 1. This reduces all belief state updates to the form of (2.14) and makes the sets Θ and B irrelevant in the model. The ability of the actions to change the state of an object remains in the POMDP, however, so we can still allocate the actions based on their costs and effects.

To handle the other source of randomness, we can make all consumptions deterministic and feasible by replacing them with their upper bounds. Combining this with the single-action modification above allows us to define a DP recursion for a related POMDP that generates deterministic policies. Suppose we are given a POMDP specified by $\langle E, A, P, R, \Theta, B\rangle$, and let $\overline{W}_i(e,a)$ be the maximum value that the random variable $W_i(e,a)$ can attain. Now define the following:

$$\overline{w}_{ia} \equiv \max_{e\in E}\{\overline{W}_i(e,a)\}; \tag{5.15}$$

$$\Theta' \equiv \{\text{null}\}; \tag{5.16}$$

$$B' \equiv \{\Pr(\theta|a,e)\colon \Pr(\theta = \text{null}|a,e) = 1\ \forall a \in A, e \in E\}; \tag{5.17}$$

$$R' \equiv \left\{\begin{matrix} \overline{r}_{ea}, r_e\colon r_e \in R,\ \overline{r}_{ea} = \sum_{i\in I^{rigid}} \lambda_i \overline{w}_{ia} + \sum_{i\in I^{soft}} \lambda_i E[W_i(e,a)], \\ a \in A, e \in E \end{matrix}\right\}. \tag{5.18}$$

As in (3.3), let $P^a \equiv [\Pr(e|a,e')]$ be the |E| x |E| matrix of transition probabilities. Then, the DP recursion (2.16) for the deterministic POMDP $\langle E, A, P, R', \Theta', B'\rangle$ reduces to the following, which we denote as $DP^L(\pi,\lambda)$:

$$\begin{aligned} DP^L(\pi;\lambda)\colon\quad & val_0[\pi,\lambda] = \sum_{e\in E} \pi_e r_e, \\ & val_n[\pi,\lambda] = \max_{a\in A}\left\{-\sum_{e\in E} \pi_e \overline{r}_{ea} + val_{n-1}(P^a \cdot \pi, \lambda)\right\}, \\ & \quad n = 1,2,\ldots T. \end{aligned} \tag{5.19}$$

We summarize the above discussion in the following lemma:

Lemma 5.4: Given the costs λ, a POMDP specified by $\langle E,A,P,R,\Theta,B\rangle$, the related POMDP $\langle E,A,P,R',\Theta',B'\rangle$ as specified by (5.15)-(5.18), and a set of rigidly-constrained resources $I^{rigid} \subseteq I$, the optimal policy *s* produced by the recursion $DP^L(\pi,\lambda)$ is admissible for the original POMDP $\langle E,A,P,R,\Theta,B\rangle$. Furthermore, s ∈ $S^D(I^{rigid})$.

Proof: The optimal policy produced by $DP^L(\pi,\lambda)$ is a map from the belief space to the actions in A and by definition is admissible for the POMDP $\langle E,A,P,R,\Theta,B\rangle$. The consumptions for the rigidly-constrained resources are fixed at their upper bounds, and the DP recursion chooses a particular action with probability 1 for any belief state and stage. Therefore, the optimal policy follows a deterministic path for any starting belief state. Since the rigidly-constrained consumptions are fixed for all actions and the actions taken are deterministic, the total consumptions are deterministic. ■

There is no uncertainty about the total consumption of any allocation of these policies because we can coerce the POMDPs into producing policies that consume deterministic amounts of the rigid resources. Let $\overline{Y}_{sji}$ be the deterministic bound on the consumption of resource i when controlling object j with policy s. Then, we can formulate an optimization that gives a feasible answer to the rigid problem:

$$MPR(S):\quad \max_x \sum_{j\in J}\sum_{s\in S_j} E\left(R_{sj}\right)x_{sj} \tag{5.20}$$

$$\sum_{j\in J}\sum_{s\in S_j} \overline{Y}_{sji}\, x_{sj} \le b_i \quad \forall\, i \in I^{rigid}$$

$$\sum_{j\in J}\sum_{s\in S_j} E\left(Y_{sji}\right) x_{sj} \le b_i \quad \forall\, i \in I^{soft} \tag{5.21}$$

$$\sum_{s\in S_j} x_{sj} = N_j \quad \forall\, j \in J$$

$$x_{sj} \ge 0 \text{ and integer } \quad \forall\, j \in J, s \in S_j .$$

We can now give the following theorem for the lower bound:

Theorem 5.5: Let $S^D\left(I^{rigid}\right) = \bigcup_{j\in J} S_j^D\left(I^{rigid}\right)$ be the set of policies that can be generated by constructing the related POMDPs $\left\langle E, A, P, R', \Theta', B' \right\rangle$ for each class *j* of objects in problem *SPR*(S^*). Then *v*[*MPR*($S^D[I^{rigid}]$)] ≤ *v*[*SPR*(S^*)].

Proof: By Lemma 5.4, the consumptions of the rigid resources for any policy produced by the related POMDPs are deterministic upper bounds on the actual resources consumed. Therefore, any allocation of these policies to objects results in deterministic upper bounds, so the optimal solution to *MPR*[$S^D(I^{rigid})$] satisfies the constraints of the rigid problem and is a feasible solution to *SPR*(S^*). Since it is a feasible solution, its value must be less than or equal to the value of the optimal solution, and *v*[*MPR*($S^D[I^{rigid}]$)] ≤ *v*[*SPR*(S^*)]. ■

The usefulness of this bound depends on the distributions of the random consumptions $W_i(e,a)$ and the overall value of information in the model. If the distributions have large variances, the policies generated for the bound always assume worst-case consumption and may be tremendously conservative. Similarly, accurate information allows the model to use resources more efficiently, and completely removing this capability may also result in a very conservative bound.

3. Bounds for the Sensor-Shooter Example

To test this lower bound with the sensor-shooter example of Chapter IV, we make all but the aircraft attrition constraint (4.6) rigid and require integral allocations. Modeling attrition as a soft constraint is realistic; when a commander says "don't lose more than 2 aircraft," he doesn't mean to fly at most 2 sorties out of a 72-aircraft fighter wing. Rather, he is communicating his tolerance for attrition.

In the sensor-shooter example, setting the available number of sensor looks to 0 in the constraints (4.5) sets their dual costs so high that the POMDPs never use them (it would be more efficient to eliminate sensor looks from the actions available to the model, but this means we lose the dual information on the looks). The resulting policies are deterministic sequences of attacks and pauses, and the only random consumption is aircraft attrition, which we allow to be soft.

To enforce the integrality requirement, note that the formulation of the sensor-shooter example has all nonnegative constraint coefficients and nonnegative constraint right-hand sides. As a result, truncating the final solution (i.e., $x'_{gs} = \lfloor x^*_{gs} \rfloor$) yields a feasible answer. We also solve the final master LP as an integer program, restricting the columns to those generated in the original decomposition. We use the CPLEX math programming package (ILOG 1997) to solve the integer program.

Table 5.1 shows the upper bound, the lower bound computed by truncation, the lower bound from the integer programming solution, and two *upper bounds* on the integer programming solution. The "B-B upper bound" is computed as a part of the branch-and-bound algorithm (e.g., Parker and Rardin 1988, pp. 159-165), and is a bound on the best possible integer solution using the subset of the columns of S available in the final master LP. The integral version of the master problem, with over 3,000 integer variables, is too large to solve to provable integer optimality with a branch-and-bound algorithm, so we terminate the run when it reaches a 2% "integrality gap" (analogous to the decomposition gap used in Chapter IV). The "best possible integer solution" is just the final upper bound produced by the decomposition, and is a bound on the best possible integer solution. The integer solution is a better measure of the lower bound, because it has an associated feasible allocation; there is no guarantee that a feasible integer solution exists that achieves the upper bounds.

Upper Bound	Lower Bounds			
	Truncated Integer Solution	B-B Integer Solution	B-B Integer Upper Bound	Best Possible Integer Solution
61185.16	55030.39	55962.57	56033.11	56130.53

Relative Gap	0.1006	0.0854	0.0842	0.0826

Table 5.1: Upper and Lower Bounds on a Rigid Version of the Sensor-Shooter Example. This table shows the bounds on the objective function of the rigid sensor-shooter problem. The upper bound is the smallest upper bound reported in Figure 4.7. The various lower bounds are computed by eliminating sensor looks from the model and considering only deterministic attack policies. The truncated lower bound comes from truncating the decomposition solution; the integer solution was computed by converting the final master LP of the decomposition to a integer program and solving using branch-and-bound (B-B). The last two columns are upper bounds on the integer solutions; the B-B bound only considers a subset of *S*, and "best" bound is the upper bound on *LP*(*S*). There is no guarantee that an integral solution exists that achieves either of these bounds.

While these bounds are not as tight as we would like them to be, they still provide useful information about the location of $v[SPR(S^*)]$. The lower bound is probably conservative, so we suspect the distribution of the total payoff is skewed towards the upper bound (we present empirical evidence that this is the case in Section V.C).

Furthermore, the difference in the bounds estimates the potential value of the information resources in the model, which is one of the principal motivating questions of this dissertation. Table 5.1 shows that adding BDA sensors improves the total expected payoff by at most 9.6%, which gives us a very useful measure of the value of information (although we again remind the reader that this is a notional data set, and should not be used to judge actual capabilities of US sensor or weapons systems).

As an aside, solving the decomposition with no sensors verifies theory covered in Section III.B.1. With only one possible observation, the total solution time for the sensor-shooter example drops from 118 to 17 seconds.

4. A Simulation for Estimating Distributions of Outcomes

Solving *LP2(S)* yields an allocation of policies to objects across a time horizon and an expected total payoff. One approach to employing the solution is to merely allocate the policies as recommended by the decomposition and follow them; if all the constraints are soft, infeasibilities are allowed and we can always get whatever resource we need.

The rigid problem $SPR(S^*)$ does not allow this. Once we run out of a rigid resource, it is simply gone, and we cannot follow a policy that calls for more of it. While we can compute the analytical upper and lower bounds of the previous section, they do not give us a complete picture of the distribution of outcomes under a rigid policy, nor do they give us much information about the policy and resource allocations.

Estimating these distributions is important. The mathematical programming community expends great effort on finding bounds on objective function values, and we have exploited this theory in our work. Nevertheless, objective function bounds do not describe the distribution of outcomes, and in allocation problems, the distributions of allocations and resource usage are often more important than the distribution of objective function values. Consider the sensor-shooter problem, where the objective function is in terms of a total value. While the individual target values are useful to determine how to allocate resources among the targets, the total expected payoff is more difficult to interpret. In practice, campaign planners are much more interested in the characteristics of the allocations that the total expected value gained.

Also, authors such as Geoffrion and Powers (1995) and Brown, Dell, and Wood (1997) have commented on the fragility of optimization solutions and their tendency to produce solutions that use resources in an extreme fashion. The randomness in the rigid problem makes the likelihood of executing an upper bound solution small, so we cannot gain much insight from the allocations produced by the upper bound computation.

To estimate the distributions, we need a way to construct a *joint* policy that satisfies the rigid problem, and our approach is to apply the decomposition sequentially across the time horizon. At the beginning of each time period, we run the decomposition to determine the policies for the rest of the time horizon and the allocations. But, we only execute the actions specified in the solution for the current time period. We specify rules to use while executing the policies to ensure we do not violate the rigid resource constraints, and update the belief states based on the observations. We then adjust the resources remaining and solve the decomposition for the next period. Note that we must assume the consumptions are completely observable; the POMDP model allows partial observability for consumptions, but we do not treat that case.

We could just follow the policies originally generated and abandon any object requiring resources that have been exhausted. Nevertheless, slavishly following policies is not reasonable when we already possess methodology to reoptimize the problem. Reoptimization makes sense from an economic point of view; once we adjust the belief states of the objects and determine the resource usage, the dual costs of the resources have probably changed completely and the old policies and allocations are probably suboptimal.

Therefore, we construct the joint policy by successively reoptimizing the problem based on the current belief states of the objects and the remaining resources. In this way, we force the allocations to react to the belief states of the objects and the remaining resources as the rigid policy would. This requires us to determine first, how to reoptimize the problem, and second, how to account for resource consumption.

At first glance, the reoptimization problem appears difficult. After we take actions, we may end up with as many as $|A|^{|\Theta|}$ different belief states for objects that all started in an identical class. The stipulation of the multiple-object algorithm of Section II.C was that all objects in each class start in the same belief state $\pi_j(1)$. If the number of states can grow exponentially for each object class, it seems that we have to solve separate subproblems for each of these new classes, making reoptimization unmanageable.

Fortunately, this is not the case. Recall that the optimal policy for a POMDP is a map from *all* belief states and time periods to the actions. The number of initial belief states is irrelevant, because the DP recursion provides a policy that optimizes for all of them. Let L_j be the set of all initial belief states for an object in class j. We now rewrite *LP2(S)* (2.18) for the case of multiple initial belief states, and denote this master LP as *LPM(S)*:

$$\begin{aligned} LPM(S): \quad & \max_x \sum_{j\in J}\sum_{s\in S_j}\sum_{l\in L_j} E(R_{sjl})x_{sjl} \\ st \quad & \sum_{j\in J}\sum_{s\in S_j}\sum_{l\in L_j} E(Y_{sjli})x_{sjl} \le b_i \quad \forall\, i\in I \\ & \sum_{s\in S_j} x_{sjl} = N_{jl} \qquad \forall\, j\in J, l\in L_j \\ & x_{sjl}\ge 0 \quad \forall\, j\in J, s\in S_j, l\in L_j. \end{aligned} \tag{5.22}$$

The computations for the upper bound (2.20) and (2.21) change as follows:

$$u_{jl}(S_j;\lambda) = \max_{s\in S_j}\left\{ E(R_{sjl}) - \sum_{i\in I}\lambda_i\, E(Y_{sjli})\right\} \equiv val_T^{jl}\left[\pi^{jl}(1),\lambda\right]; \tag{5.23}$$

$$u(S;\lambda) = \sum_{i\in I}\lambda_i b_i + \sum_{j\in J}\sum_{l\in L_j} N_{jl}\, u_{jl}(S_j;\lambda). \tag{5.24}$$

LPM(S) requires exactly the same amount of work in the subproblems as *LP2(S)*. The only substantial difference in the two master problems is the expansion of the allocation constraints to allow for each combination of starting belief state and object class. The number of these constraints can potentially grow large, but we have room for growth in the master LP.

LPM(S) has some advantages over *LP2(S)*. The POMDPs can now produce up to $|L_j|$ improving columns in each iteration, so much more information is passed between the master problem and the subproblems. Also, many of the objects may simply be ignored in the problem. For example, if we have a target in the sensor-shooter problem with an initial belief state of 0.99, it is almost surely dead. Unless the target is extraordinarily important or we have excess resources, the master LP will not allocate resources to it.

We also present a mixed-integer version of *LPM(S)* to solve for integral assignments for the current time period ($t = 1$). In any policy, there is no uncertainty about the action taken in the first time period (i.e., the stage with T stages to go). Since we are trying to determine which actions to take immediately, we only need an integral allocation of the policies whose actions potentially consume resources in the first time period. An example of this is the sensor-shooter example; any policy that pauses in time period 1 cannot consume any resources, and doesn't require a random allocation. This reduces the number of integral variables in the model.

Let $S_1 \subseteq S$ be the subset of policies that can consume any resource in the first time period. Then, the integral allocation model *MPM(S)* is identical to *LPM(S)* except for the following condition:

$$x_{sjl} \text{ integral} \quad \forall\, j \in J, s \in S_j \cap S_1, l \in L_j. \tag{5.25}$$

In this dissertation, we do not use the decomposition to solve *MPM(S)*. Instead, we restrict the policies (columns) to those generated by *LPM(S)*. We denote this set by *S′*, so we actually solve *MPM(S′)* rather than *MPM(S)*. Solving *MPM(S)* would require embedding the decomposition within a branch-and-bound procedure (e.g., Barnhart et al. 1998). Table 5.1 shows a particular case where the integer solution to *MPM(S′)* is within 0.26% of *LPM(S)*. Such small integrality gaps are typical in the sensor-shooter example, and since gaps of that magnitude are generally acceptable in mixed-integer programming, we have not refined our approach any further.

We also require rules on resource consumption for both rigid and soft resources. If we allocate actions that consume random amounts of rigid resources, we must define the order in which policies are executed within a period. Otherwise, we have no way of knowing which policies cannot be executed when some resource runs out. Observing the random consumptions and then deciding which policies to execute violates the dynamics of the POMDP model as we have defined it in Section II.B.5. Therefore, the rule cannot be based on the realization of the consumption. On the other hand, if all policies to be used have deterministic consumptions, they must meet the constraints and the order of allocation is irrelevant.

We also must decide how to handle soft resources. Suppose we allow expected losses of 3 aircraft of a particular type across a 9-period horizon in the sensor-shooter example. We suggest two alternatives of how to enforce this constraint:

- **Convert the resource to a "use or lose" resource available in each period.** We change the master LP to constrain the expected consumption by spreading it over each period. In the example, we would put in an attrition constraint for every period that limits attrition to 1/3 of an aircraft each period. The decomposition plans accordingly, and the expected losses will be less than or equal to three.

- **Update resources, but allow violations within one period.** We make all of the resource remaining available to each successive decomposition. For example, suppose we lose two aircraft of a particular type in the first period. We would then reduce the available attrition in the second period to one. If we lose one or more aircraft in the second period, that aircraft type is not be available for the rest of the periods. This method will tend to result in expected consumptions higher than the original constraint, due to the possibility of overshooting within a period.

We can now outline a simulation procedure to generate rigid joint policies. The simulation takes advantage of the fact that we can reoptimize at the start of each time period and generate new policies based on resource consumption and the belief states of the objects. Let *REP* be the number of simulation repetitions desired, *T* be the number of time periods in the horizon, and *k* and *n* be index counters. Then, the general procedure is as follows:

1. **Set $k = 0$.**
2. **If $k = REP$, stop; otherwise, set $k = k + 1$, reset each object to its initial belief state, and set $n = T$.**
3. **Solve *LPM(S)* for the current states of all the objects, the current resources available, and *n* stages yielding the set of policies S'.**
4. **Solve *MPM(S')* to get integral policy assignments for the current stage.**
5. **Simulate (or observe) outcomes for the current stage and update object states and resources used.**
6. **If $n = 1$, record this set of results and go to 2; otherwise, set $n = n - 1$ and go to 3.**

The simulation has value far beyond estimating the mean reward of the simulated joint policy. We can use it to investigate the distributions of any random outcome in the model, which is something we cannot derive from the analytical bounds.

B. SIMULATING THE AVIV-KRESS TARGETING PROBLEM

1. The Aviv-Kress Model

Aviv and Kress (1997) analyze several different "shoot-look-shoot" tactics for a model of a single shooter engaging *m* identical targets. In their scenario, they assume the shooter can take a total of *n* shots, each of which has a probability of kill *pk*. Following each shot, the shooter looks at the target, and given the target is actually dead, assesses it as dead with probability *psk*. The shooter does not mistake live targets for dead, so the assessment can only make one type of error.

Each shoot-look sequence is an independent trial, and looks only occur in conjunction with a shot. Finally, the shooter is invulnerable.

The authors measure the effectiveness of the tactics by the expected number of targets killed. For this scenario, Manor and Kress (1996) prove that the optimal tactic is the so-called "greedy shooting" strategy. Greedy shooting means to allocate the next shot to the least-attacked target that still appears alive. Aviv and Kress develop and assess other tactics because greedy shooting requires labeling each target, remembering the number of shots fired at each target, and remembering the latest assessment. The authors comment that this requirement is difficult to meet in ground combat, as the constant redirection of fire would hurt combat effectiveness (Aviv and Kress 1997, p. 87).

Aviv and Kress develop expressions for expected kills for the various tactics by defining the target states as either "live," "evidently killed," and "killed but assessed as live," and derive the expectations of the number of targets in each state directly using probability theory. They do not model sensor delays or reliabilities, nor do they allow sensor looks to occur without an accompanying shot. They also note that the models presented do not work if the sensors can assess live targets as dead, and offer that case as an area for future research.

2. Analytical Solution and Implementation Using the Decomposition

The Aviv-Kress model fits the framework of the general problem in this dissertation, and is in fact a special case of the sensor-shooter problem. There are two available actions (pause and shoot-look, or *sl*), and *n* time intervals. We can take at most one shoot-look in each time interval, and we must allocate that shoot-look to one of the *m* targets. The targets have two states (live or dead) and the states are partially observable. The sensor has two possible observations (live or dead), has a response probability of 1 and a response time of 1. By setting the value of each dead target to 1, the objective function counts the expected number of targets killed. Since there is only a single resource, we can eliminate all but the sensor look (4.6), allocation (4.4), and nonnegativity constraints (4.7) in the master LP.

The only modification required to the sensor-shooter model necessary is to define a shoot-look action. This requires changing the updates (4.12) and the conditional probabilities (4.13):

$$\pi' \equiv \pi + pk(1-\pi); \tag{5.26}$$

$$\begin{aligned} Tr(\pi| sl, \theta = \text{Live}) &= \frac{(1-psk)\pi'}{(1-\pi)+(1-psk)\pi'}; \\ Tr(\pi| sl, \theta = \text{Dead}) &= 1; \end{aligned} \tag{5.27}$$

$$\begin{aligned} \Pr(\theta = \text{Live}| \pi, sl) &= (1-\pi') + (1-psk)\pi'; \\ \Pr(\theta = \text{Dead}| \pi, sl) &= psk \cdot \pi'. \end{aligned} \tag{5.28}$$

The Aviv-Kress model is a good test case for the simulation developed in Section V.B.2, because it is a rigid problem, has been examined in the literature using other methods, and it has an analytical solution. To compute the analytical solution, we develop a DP algorithm to find the expected kills using greedy shooting. As above, let n be the total shots available, t be the number of shots left, pk be the probability of killing the target, and psk be the probability of assessing a dead target as live. Let m be the number of targets available, x be the number of targets alive, and y be the number of targets "evidently-live" (targets both alive and assessed as alive). The state space of this process is then $\{(x, y): \quad x = 0,1,\dots, y, y = 0,1,\dots m\}$.

The greedy shooting strategy divides the entire engagement into "rounds," with each round consisting of firing one shot at each evidently live target. After every round (except for the last one), all remaining evidently live targets have been attacked an equal number of times. Let X be the random number of surviving targets, and Y be the random number of apparently surviving targets. The next equation gives the state transition probabilities if the current state is (*x*,*y*) and $t > y$:

$$\Pr(X = i, Y = j | x, y) \\ = \begin{cases} \left[\binom{x}{x-i} pk^{x-i}(1-pk)^{i}\right]\left[\binom{y-i}{j-i} psk^{y-j}(1-psk)^{j-i}\right], \\ \quad x \geq i, y \geq j, i \geq j, j \leq m; \\ \\ 0, \text{ otherwise.} \end{cases} \tag{5.29}$$

The first bracketed term is the probability that *i* targets survive the current round of shots, and the second bracketed term is the probability that *j-i* targets are dead and assessed as live.

If $t \leq y$, we must allocate the remaining shots randomly among the *y* evidently-live targets. Define $F_t(x, y)$ to be the expected targets surviving with *t* stages left and current state (*x*,*y*). If we do not have enough shots to cover all evidently-live targets, we will choose each live target with probability *x*/*y* and kill it with probability *pk*. As a result,

$$F_t(x, y) = x - \frac{x \cdot t \cdot pk}{y}, t \leq y. \tag{5.30}$$

If we have enough shots to cover the remaining evidently-live targets, the next equation gives the total expected kills when we are in state (*x*,*y*) and have *t* shots left:

$$E\left[F_{t-y}(x, y)\right] = \sum_{i=0}^{x}\sum_{j=i}^{y} \Pr(X = i, Y = j | x, y) F_{t-y}(i, j), \\ t \geq y. \tag{5.31}$$

Then the following recursion computes the expected targets surviving under the greedy shooting policy:

$$F_t(x,y)=\begin{cases} x, & t=0 \text{ (no shots left);} \\ 0, & x=0 \text{ (all targets killed);} \\ (1-pk)^t, & x=1, y=1 \text{ (1 target left);} \\ x-\dfrac{x\cdot t\cdot pk}{y}, & y\ge t; \\ E\left[F_{t-y}(x,y)\right], & y<t. \end{cases} \tag{5.32}$$

3. Sample Results

If this scenario is really an instance of our general problem, then the decomposition should produce an upper bound on the analytical result computed by (5.32). To test this theory, we compute the exact greedy shooting result for 4 cases cited by Aviv and Kress, and run the decomposition for the same cases. We also constructed a simulation as outlined in Section V.A.4, and ran each of the 4 cases for 500 trials to analyze the distributions of the expected kills.

Probability of kill (pk)	Probability of assessing a dead target as dead (psk)	Analytical expected kills	Decomposition expected kills
0.5	0.5	8.10	8.10
0.2	0.9	3.95	3.95
0.2	0.1	3.63	3.63
0.8	0.1	9.63	9.63

Probability of kill (pk)	Probability of assessing a dead target as dead (psk)	Simulation sample mean	Confidence Intervals	
			99% lower bound	99% upper bound
0.5	0.5	8.15	8.06	8.23
0.2	0.9	3.95	3.82	4.09
0.2	0.1	3.64	3.58	3.70
0.8	0.1	9.63	9.62	9.64

Table 5.2: Comparison of the Decomposition, the Decomposition Simulation, and the Analytical Expected Kills for the Aviv-Kress Model. The top table shows that the decomposition solution matches the analytical solution in all 4 cases. The bottom table shows that a 99% confidence interval on the simulation mean contains the analytical mean in each case. The simulations were run with 500 repetitions; each model case consists of 10 targets and 20 shots.

We show the results in Table 5.2. In each case, the decomposition terminates with an error gap of zero and matches the analytical answer given by (5.32), so the decomposition computes the exact analytical answer and not just an upper bound. Also, we report the sample expected number of kills from the simulation and the 99% confidence interval on the mean. In each case, the analytical mean falls within the confidence interval. The runtime for the decomposition for this problem was less than 20 seconds in all cases.

We can also compare the sample distribution of target kills generated by the simulation with the exact distribution. Figure 5.1 shows a sample of 500 repetitions of the simulation compared to the exact distribution of target kills. In this particular case, $pk = 0.5$, $psk = 0.5$, $m = 10$, and $t = 20$. The sample distribution follows the exact distribution closely, which may be a bit surprising considering that the simulation does not allow randomized allocations of policies. Using a chi-square goodness-of-fit test (e.g., Larsen and Marx 1986, pp. 402-406), we cannot reject the null hypothesis that the distributions are identical. The p-value (smallest level of significance for which the null hypothesis can be rejected) for the test is 0.264.

More importantly, this small-scale example demonstrates that combining the decomposition with a simulation can produce accurate sample distributions. While the simulation isn't really necessary for the greedy shooting model, we stress that we cannot compute the exact distributions for most models of interest. Aviv and Kress expend considerable effort to obtain analytical results, but their model is limited to a single shooter, a single target type, a single sensor, and a single type of assessment error. The decomposition approach we have developed eliminates all these limitations, but can only bound an underlying rigid problem. In the next section, we show a similar analysis for the sensor-shooter example.

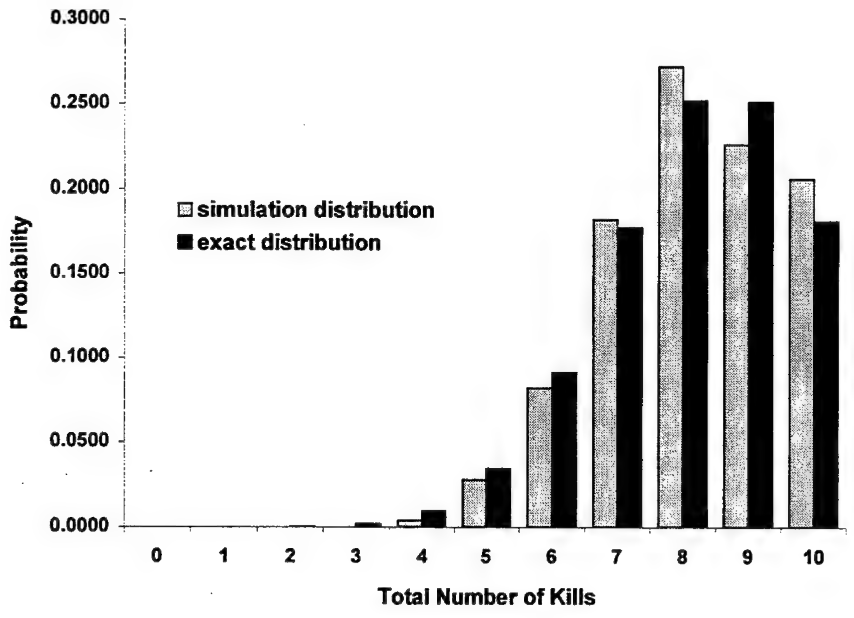

Figure 5.1: Comparison of the Exact Distribution of Killed Targets to a Sample Distribution from the Simulation. This chart shows the exact distribution of kills for the greedy shooting strategy with *pk* = 0.5, *psk* = 0.5, 20 available shots, and 10 targets. The sample distribution was constructed from 500 repetitions of the decomposition simulation, and closely matches the exact distribution.

C. SIMULATING THE SENSOR-SHOOTER EXAMPLE

1. The Rigid Sensor-Shooter Model

To use the simulation procedure of Section V.A.4, we must decide what rigid version of the sensor-shooter example to solve. Except for aircraft attrition, the consumptions in the model for each action are fixed, so the only randomness for those resources is whether or not a particular policy has to take the action. As a result, we enforce rigid constraints on aircraft sorties, weapon usage, and sensor looks. We also do not allow randomized allocations, so we must create integer allocations in each time period. Note also that we assume sortie availability is unaffected by attrition; that is, we assume that losses will be low enough that the remaining aircraft can "surge" and still provide the sorties.

Our rigid version of the sensor-shooter problem allows attrition to be soft, but the rule we use when the simulated attrition meets or exceeds the soft constraint modifies the previous definition. For the analyses shown here, we allow attrition to exceed the constraint *within* the time period. This is equivalent to assuming all sorties are launched at the start of the period, so there is no opportunity to recall any aircraft if the losses get too high too quickly.

Once the period ends and we see that the attrition for a particular aircraft type is at or over the limit, we do not fly that aircraft for the rest of the time horizon. There is functional justification for this assumption. Cohen (1993, p. 279) reports an 8-hour period during DESERT STORM in which one A-10 attack aircraft was severely damaged by a surface-to-air missile and two others were shot down. The immediate reaction was to restrict the aircraft from operating in heavily-defended areas, which had the same effect as cutting them off from the bulk of the available targets. These restrictions were not removed until the start of the ground war 10 days later.

Unfortunately, we cannot formulate *MPM*(*S*), the mixed-integer version of the master problem, to enforce the rule above. This negates the lower bound computed in Table 5.1, which is based on having a soft constraint for attrition throughout the time horizon. Nonetheless, we present this heuristic lower bound in this section for comparison.

The formulation for *LPM*(*S*), the sensor-shooter example with targets in multiple belief states, requires two modifications from the formulation presented in Section IV.A.2. First, we must add an index $l \in L$ for targets in various starting belief states, and second, we must modify the objective function to account for the starting states. Define CS_{lg} as the current belief state $l \in L$ for a target of type g. Then, the objective function is

$$\max_x \sum_{g\in G} \sum_{s\in S_g} \sum_{l\in L} V_g \left(PD_{gsl} - CS_{gl}\right) x_{gsl} + \sum_{g\in G} \sum_{l\in L} V_g \, TGT_{gl} \, CS_{gl} \,. \tag{5.33}$$

The constraints are:

$$\sum_{g\in G} \sum_{s\in S_g} \sum_{l\in L} ES_{igslt} \, x_{gsl} \le SORT_{it}, \quad \forall \, i,t \quad \left(sd_{it}\right); \tag{5.34}$$

$$\sum_{g\in G} \sum_{s\in S_g} \sum_{l\in L} EW_{wgsl} \, x_{gsl} \le WPN_w, \quad \forall \, w \quad \left(wd_w\right); \tag{5.35}$$

$$\sum_{s\in S_g} x_{gsl} = TGT_{gl}, \quad \forall \, g,l \quad \left(td_{gl}\right); \tag{5.36}$$

$$\sum_{g\in G} \sum_{s\in S_g} \sum_{l\in L} EL_{ogslt} \, x_{gsl} \le LOOK_{ot}, \quad \forall \, o,t \quad \left(ld_{ot}\right); \tag{5.37}$$

$$\sum_{g\in G} \sum_{s\in S_g} \sum_{l\in L} EA_{isl} \, x_{gsl} \le MAXATT_i, \quad \forall \, i \quad \left(ad_i\right); \tag{5.38}$$

$$x_{gsl} \ge 0, \qquad \forall \, g \in G, s \in S_g, l \in L \,. \tag{5.39}$$

In formulating the model *MPM(S)*, we only require integral allocations for policies which take actions in the first time period. The consumptions $W_i(e,a)$ are fixed for the rigid resources and independent of the target state in the sensor-shooter model, and the action taken by any policy in the first time period is deterministic. Therefore, any feasible allocation of policies meets the rigid constraints associated with the first time period with probability 1. Furthermore, we require integral allocations only for policies that expend resources in the first period.

To construct *MPM(S)*, we only need to revise the constraints in (5.39). As in Section V.A.4, let $S_1 \subseteq S$ be the subset of policies that can consume any resource in the first time period. Then the integrality condition can be written as

$$x_{gsl} \text{ integral } \forall s \in S_1 \,. \tag{5.40}$$

2. Accelerating the Simulation

The simulation of the sensor-shooter example capitalizes heavily on the reductions in runtime realized in Chapter IV. Nonetheless, running the simulation for 150 repetitions requires almost four hours of computing time on our system.

We employ a few tricks to speed up the simulation. Since the initial *T*-period integral allocation is not random, we only need to compute it once and save the resulting policies. Now, the simulated outcomes from the allocation differ among repetitions, but drawing random numbers and storing results consumes very little overhead. As a result, we only need to solve *T*-1 decompositions for every repetition. For our 9-period example, this eliminates 150 decomposition solutions.

Also, we reuse applicable policies generated in the decomposition solution for an *n*-stage problem in the solution of the *n*-1st stage decomposition. Any policy that pauses in the first period of the *n*-stage model could be used directly in the *n*-1 stage model, so these policies replace the heuristic used in Section IV.A.7 for generating initial policies. These policies tend to be "better" than those generated by the heuristic, and they accelerate the subsequent decomposition solutions.

We also do a quick check to see if truncating the solution from *LPM*(*S*) meets the integrality gap tolerance. If so, we have an immediate integral allocation and do not use branch-and-bound to improve the solution. Even if the truncated solution does not meet the gap, we use it as an "incumbent solution" in the branch-and-bound algorithm to accelerate the solution of *MPM*(*S*). We can limit the maximum time we allow the branch-and-bound procedure without any danger of terminating without a feasible solution, because truncating yields a feasible incumbent solution.

Finally, we use a looser gap tolerance (0.01) for the decompositions in the simulation to accelerate the overall solution time. This change moderates the solution time spent in the "tail" of the decomposition, when the algorithm is largely trying to reduce the upper bound. While this naturally reduces the lower bound objective function values, the reduction is negligible.

3. Sample Results

We begin by examining the distribution of the optimal objective function value. Figure 5.2 shows the distribution of the objective function values for 150 repetitions of the sensor-shooter simulation. The figure empirically verifies the upper bound on the expected objective function value, but the sample distribution also provides information on how often we can exceed that bound.

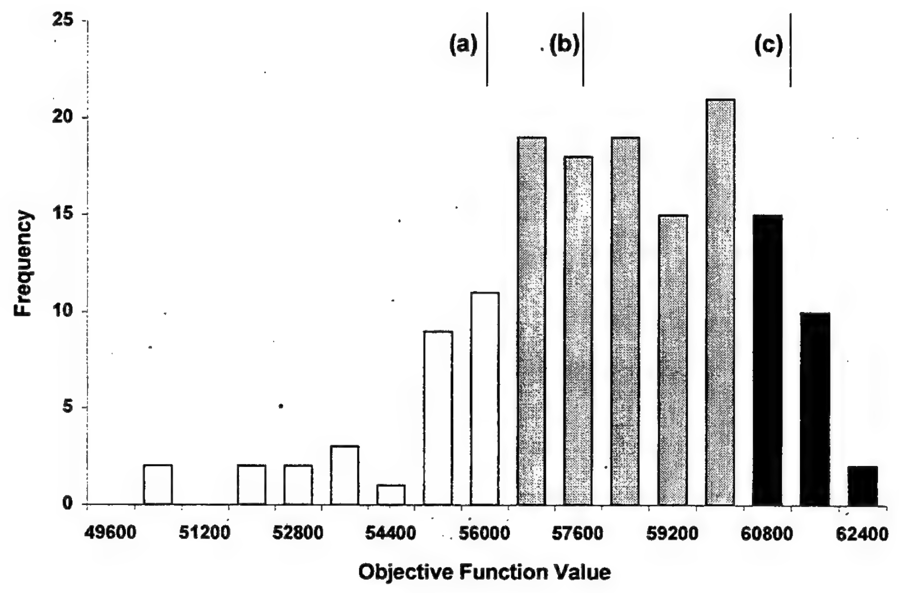

Figure 5.2: Sample Distribution of Objective Function Values from the Sensor-Shooter Simulation Example. This chart shows the sample distribution from 150 repetitions of the sensor-shooter simulation. The white bars indicate repetitions whose objective function value is lower than the lower bound on the expected objective function value (a); black bars indicate repetitions whose objective function value is higher than the upper bound on the expected objective function value (c). The sample expected objective function value (b), is within 6% of the upper bound.

As we mentioned previously, the lower bound does not really apply. Nonetheless, we show this heuristic lower bound by way of comparison, and note that it is still below the mean of the sample distribution.

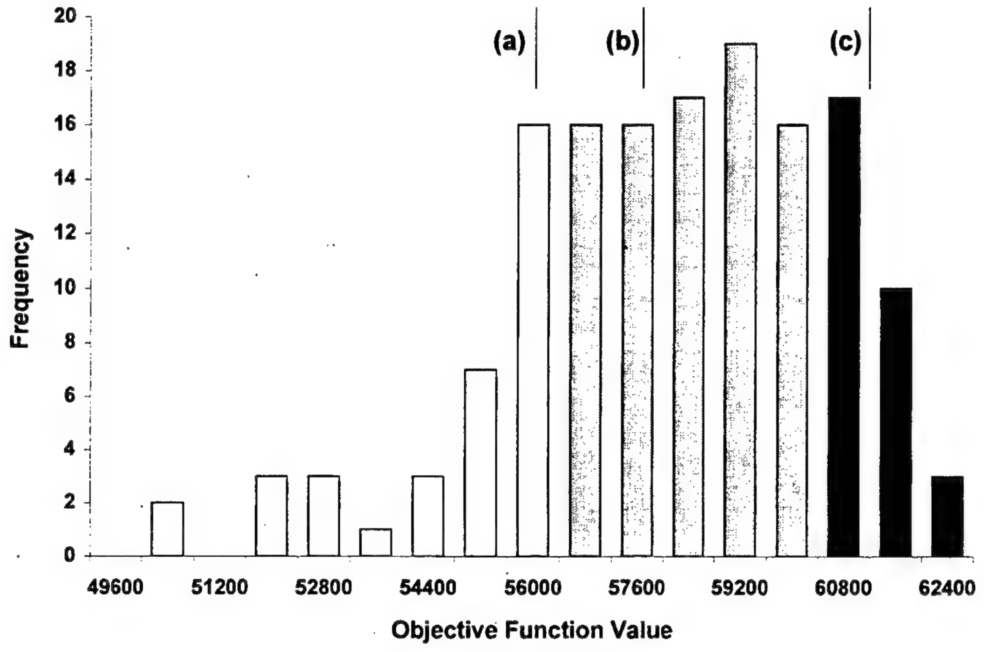

Figure 5.3: Sample Distribution of the Value of Targets Actually Killed from the Sensor-Shooter Simulation Example. This chart shows information similar to Figure 5.2, but gives the sample distribution of the value of targets actually killed, instead of the value as computed by the belief states of the targets. This distribution has a slightly lower sample mean and more variation than the distribution in Figure 5.2. (a), (b), and (c) show the lower bound on the expected value killed, the sample mean, and the upper bound, respectively.

Figure 5.2 shows the expected objective function values in terms of the final belief states in the model. In Figure 5.3, we show the sample distribution of the value of targets actually killed; in other words, the true state of reality. As we would expect, the distribution of the value of actual outcomes has a larger sample standard deviation (2467.99 versus 2412.47) than the distribution of the value of the belief states.

Both these figures support our speculation in Section V.A.3 that the distribution of objective function values is skewed towards the upper bound.

We can also investigate distributions of other random outcomes. In the sensor-shooter simulation, we record virtually all the random outcomes, so each run results in a very large (on the order of 370,000 records) database we can mine for information. We offer the following example using this database.

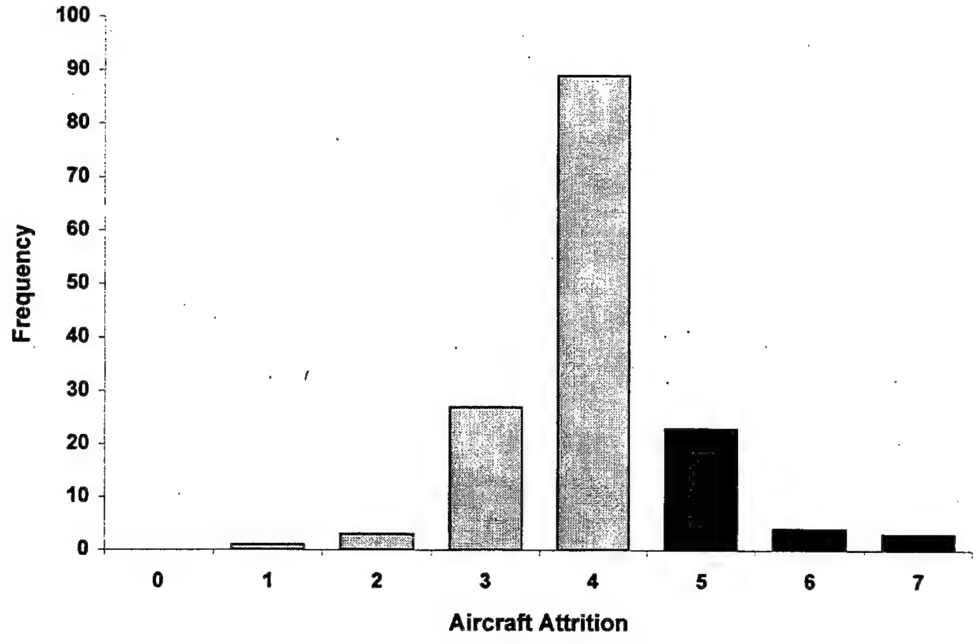

Figure 5.4: Sample Distribution of Attrition for One Aircraft Type in the Sensor-Shooter Simulation. The bars shown in black indicate repetitions where the total attrition for this aircraft type exceeded the limit of 4. While the sample mean attrition was 4.0, 20% of the repetitions experienced attrition exceeding the soft constraint value.

One question that concerns us from both a functional and an analytical point of view is the effects of the soft constraints in the model. Since our rule allows the attrition constraint to be violated in any particular time period, we want to see how often that occurs and look at the magnitude of the violations.

Figure 5.4 shows the distribution of actual attrition for an aircraft type limited to losses of 4 aircraft on the average. The losses range from 1 to 7 aircraft, but the distribution is firmly centered on the constraint right-hand-side value of 4.

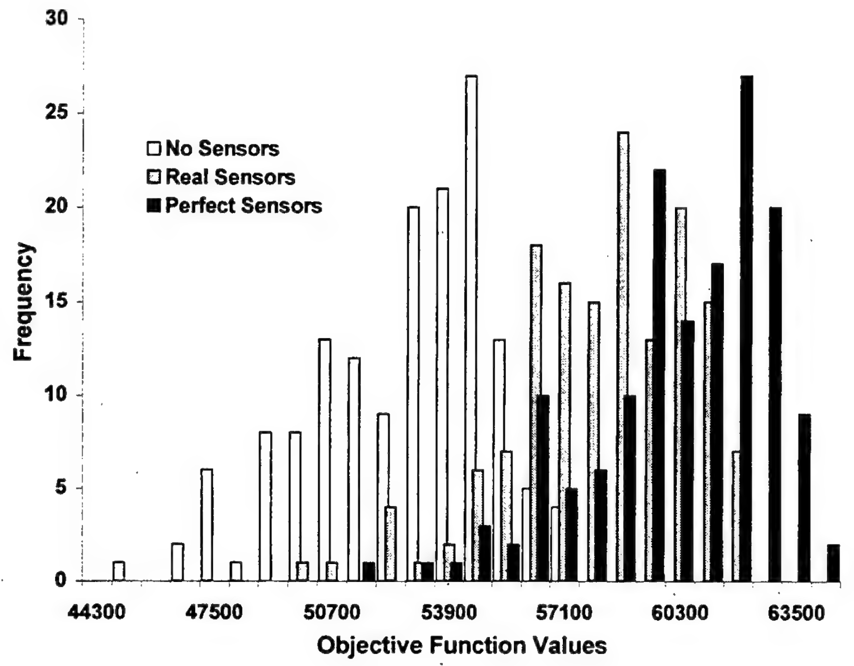

Figure 5.5: Distributions of Sample Objective Function Values for the Sensor-Shooter Simulation. This figure compares the distributions of objective function values for the sensor-shooter example with no sensors, realistic sensors, and perfect sensors. This figure shows the value of BDA sensors across the entire sample distribution of outcomes.

The final chart we offer in this section is another example of the insights that can be provided by the simulation; more importantly, it provides insights into the questions asked about the value of information in Chapter I. Figure 5.5 shows the sample distributions for the sensor-shooter example for the baseline data, a set of runs with no sensors, and a set of runs with perfect (error-free) sensors. This figure gives much more information than simply comparing the means.

Figure 5.5 also points out the ability of the decomposition to measure not just the value of perfect information, but the value of information with errors. We could use the simulation to produce similar charts for various proposed sensor types, allowing us to evaluate the overall contributions of improving BDA sensor reliability, response time, and error rates.

We make one final point about the empirical results of this section. In the stochastic programming literature, a model with rigid constraints is known as a "FAT" model (e.g., Kall and Wallace 1994, p. 8), and is rarely modeled due to the conservative nature of the solutions. The outcomes of the simulation appear to conflict with generally-accepted notions about solutions to models with FAT constraints. The simulation results show that the decomposition policies consistently result in outcomes that compare favorably to the upper bound, when we might have believed beforehand that the solutions would be very conservative.

Nevertheless, the random consumptions produced by the decomposition are very different from the random realizations of constraint parameters in a general stochastic program. The POMDP subproblems generate policies with the full knowledge of the underlying stochastic processes, and plan accordingly using the dual resource values provided by the master LP. The resulting situation violates the assumptions of the typical recourse model, where the allocations cannot affect the random outcomes.

In the decomposition, the POMDPs produce policies that are completely dependent on the expected resource costs and payoffs, but this is actually an advantage. The empirical results of this section show that passing the resource costs to the POMDP affects their distributions "favorably" in the context of the entire model, allowing the joint performance of the policies to come close to the computed upper bound.

VI. SUMMARY OF CONTRIBUTIONS AND FUTURE RESEARCH AREAS

1. Contributions to the Theory

In this dissertation, we have developed a decomposition approach for large-scale allocation problems with partially-observable outcomes. In doing so, we have used two previously-uncombined methods — linear programming and partially observable Markov decision processes — to solve a problem that is intractable using other approaches. Without combining them, we cannot solve the applicable LP problem because it has too many columns (policies), and we cannot solve the POMDP because it has too many states.

Theorem 2.6 is the key result; it shows that solving a Markov decision process (MDP) also solves the column-generation problem in the decomposition. By using the available theory, we can convert the POMDP subproblems to fully-observable MDPs and use them as subproblems in the decomposition. With the master LP providing prices on resources and the POMDP subproblems producing improving policies, the decomposition takes advantage of the strengths of each method. We show the resulting algorithm is optimal and finite, enabling us to find optimal solutions to a class of problems previously untreated in the literature. This methodology is our primary contribution.

We survey the available solution methods for POMDPs, and argue for choosing an approximate, rather than an exact, algorithm to use in the decomposition. The approximate algorithms consist of the Cheng's linear support algorithm and the grid methods; the former allows solving to a specified accuracy, while the latter allows solving with a specified amount of computational work. We choose the linear support algorithm for our empirical examples, and analyze it in detail. We present a new result on the linear support algorithm that tightens the upper bound on the optimal value function (Corollary 3.5). We also provide a formal proof for the maximum error in the value function when using the algorithm (Theorem 3.4).

We also present computational advice on implementing the decomposition, and demonstrate the improvements empirically using a large-scale military allocation problem. We develop three controls for improving the solution time of the decomposition, and succeed in reducing the total runtime for our large-scale example by almost 95%.

Finally, we extend the decomposition to a problem with resource constraints that must hold in all cases (the so-called rigid problem), and show that the decomposition can be used to compute both upper bounds (Theorem 5.3) and lower bounds (Theorem 5.5) on the objective function value. We also develop a simulation to estimate the distributions of random outcomes in the model. We demonstrate both the bounds and the simulation results for a problem in the literature with a known analytical solution (the Aviv-Kress targeting problem) and the sensor-shooter example. The decomposition solves the first problem exactly and provides upper bounds we estimate to be within 5% of the true expected optimal objective function value for the second problem.

2. Potential Applications

The empirical results that we present for the sensor-shooter example offer strong evidence of the utility of the decomposition for other problems of this type. While our motivating problem is a military application, there are many other areas where actions that produce information must compete with actions that influence the states of objects. One example in the literature is Jonsbraten (1997), who constructs a model of a well-drilling application with partial observability using a decision tree with constraints. Another example (Raffensburger 1998) concerns allocating resources to families. The objective is to minimize the incidences of spousal and child abuse, and the local community has limited resources available (social worker visits, counseling, financial aid) and the families may be grouped into various classes depending on their histories. Applying any of the resources results in a partially-observable outcome, and the families themselves may transition from violence to non-violence based on the history of actions taken and their underlying group. The decomposition could potentially reduce the number of incidents through better allocations of the limited resources.

Another application is long-term financial planning. Suppose we want to contribute to a retirement fund in the short term. We can choose an investment vehicle and invest, we can spend some money and hire a financial planner to make our investments, or we constantly reorganize our portfolio based on current results. The results of all actions are only partially observable in the short term; we won't know the final value of our portfolio until we retire.

We again emphasize this method applies to problems whose outcomes are MDPs and are completely observable. For example, the problem presented by Meleau et al. (1998) on allocating aircraft sorties and weapons to targets is a special case of the sensor-shooter example, and we can solve it with the decomposition and the simulation. Indeed, if the "core problem" for each object can be solved using stochastic dynamic programming, the decomposition can solve the overall allocation problem with shared resources.

3. Future Research

For the decomposition to run quickly, we must be able to solve the POMDP subproblems quickly. In the sensor-shooter example, we can use a one-dimensional state space for each target and greatly accelerate the solutions. For objects with more states, the POMDPs could not be solved as easily by the linear support algorithm in its current form.

Conversely, the grid algorithms can solve the POMDPs with a known amount of computational effort. The question that must be answered is how to adjust the grid beforehand to provide the smaller errors needed by the decomposition as it converges. Another interesting area for research is whether we can modify the grid based on the solutions from previous iterations in the decomposition. Each region of the belief space may not require the same fidelity in the grid, particularly if a single action is optimal for large regions.

In addition, it may be possible to combine the linear support algorithm and the grid algorithms in some fashion. The linear support algorithm is essentially a variable grid scheme; recall also that Cheng (1988, pp. 76-77) suggests a variant that limits the number of vectors found. This latter scheme bounds the work done in the algorithm, at the cost of being able to specify the error beforehand. Nonetheless, it seems that there should be a straightforward way to combine the strengths of both methods.

The decomposition also may offer a way to reduce the number of object states for certain types of models. For example, suppose we expand the sensor-shooter example so that each target has three states: undetected, detected and live, and detected and dead. Instead of expanding the POMDPs for each target, we could solve a separate subproblem to determine the number of targets detected (perhaps via a POMDP) and pass the expected number of detected targets through the master LP via a constraint to the current subproblems. Such an approach might also allow us to separate subproblems that have mixes of perfectly-observable states (solvable via an MDP) and partially-observable states.

By restricting our analyses to undiscounted POMDPs, we have not explored a popular technique in the MDP literature. We have not investigated how to interpret the discounting of future rewards and cost when the POMDP is a subproblem in a decomposition; our convergence proofs do not work if we use a discount factor. We speculate a master LP could be structured in such a way that the applicable POMDP subproblem would be discounted, but we do not know what insights we might gain from such a model.

We have not adopted the view of stochastic programmers to discuss our general problem. The rigid problem is a stochastic program, and at first glance it appears that the appropriate model to use is a multistage stochastic program with recourse (e.g., Kall and Wallace 1994, pp. 26-27). Unfortunately, models of this type are generally not solvable unless the problem has a special structure (Higle 1998, Wallace 1998). Also, recourse models assume that outcomes resulting from decisions are completely observable, and that decisions do not affect the probability distributions of the random parameters in the model. Neither assumption is true in our general problem. While there are some exceptions in the stochastic programming literature (e.g., Johnsbraten, Wets, and Woodruff (1997) discuss the case where decisions can affect the distributions of the random parameters), we have found that the stochastic programming methods available do not apply to our general problem.

Nonetheless, the dynamics of the POMDP and the stochastic program with recourse are the same: we make a decision, we observe realizations of random outcomes, and we make further decisions at some cost. Our use of expected values in our constraints in the master LP (the so-called "mean-value" model) is a practice frowned upon in the stochastic programming literature. Yet, our empirical results show that the decomposition yields results much closer to an analytical upper bound than results of illustrative mean-value examples offered in stochastic programming texts (e.g., Kall and Wallace 1994, pp. 137-144). We speculate that this is due to policies computed in the POMDPs, which take recourse into account. A formal analysis of these relationships would probably be a significant contribution to the stochastic programming literature.

LIST OF REFERENCES

Aviv, Yossi, and Moshe Kress, "Evaluating the Effectiveness of Shoot-Look-Shoot Tactics in the Presence of Incomplete Damage Information," *Military Operations Research,* Vol. 3, No. 1 (1997), pp. 79-90.

Barnhart, Cynthia, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsburgh, and Pamela H. Vance, "Branch-and-Price: Column Generation for Solving Huge Integer Programs," *Operations Research,* Vol. 46, No. 3 (1998), pp. 316-329.

Bazarra, Mokhtar S., C. M. Shetty, and Hanif D. Sherali, *Nonlinear Programming: Theory and Algorithms*, John Wiley and Sons, New York, 1993.

Bazarra, Mokhtar S., John J. Jarvis, and Hanif D. Sherali, *Linear Programming and Network Flows*, John Wiley and Sons, New York, 1990.

Bellman, Richard E., *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

Bertsekas, Dimitri P., *Dynamic Programming and Stochastic Control*, Academic Press, New York, NY, 1976.

Brown, Gerald G., Dennis M. Coulter, and Alan R. Washburn, "Sortie Optimization and Munitions Planning," *Military Operations Research*, Vol. 1, No. 1 (1994), pp. 13-18.

Brown, Gerald G., Glenn W. Graves, and Maria D Honczarenko, "Design and Operation of a Multicommodity Production/Distribution System Using Primal Goal Decomposition," *Operations Research,* Vol. 33, No. 11 (1987), pp. 1469-1480.

Brown, Gerald G., Robert F. Dell, and R. Kevin Wood, "Optimization and Persistence," *Interfaces*, Vol. 27, No. 5 (1997), pp. 15-37.

Cassandra, Anthony R., Michael L. Littman, and Nevin L. Zhang, "Incremental Pruning: A Simple, Fast Exact Method for Partially Observable Markov Decision Processes," in *Proceedings of the Thirteenth Annnual Conference on Uncertainty in Artificial Intelligence (UAI-97),* Providence, RI, 1997.

Cassandra, Anthony R., *Optimal Policies for Partially Observable Markov Decision Processes,* Technical Report CS-94-14, Brown University, Providence, RI, 1994.

Castenon, David A., "Approximate Dynamic Programming for Sensor Management," in *Proceedings of the 36th IEEE Conference on Decision and Control, Vol. 2*, IEEE Control Systems Society, Danvers, MA, 1997, pp. 1202-1207.

Cheng, Hsien-Te, *Algorithms for Partially Observable Markov Decision Processes*, Ph.D. thesis, University of British Columbia, 1988.

Cohen, Eliot, ed., *Gulf War Air Power Survey, Vol. II, Operations and Effectiveness*, Department of the Air Force, Washington, D.C., 1993.

Cotsworth, William L., *The Conventional Targeting Effectiveness Model (CTEM) User's Manual,* AEM Services, Inc., S-93-020-DEN, 1993.

Dantzig, George B., and Mukund N. Thapa, *Linear Programming I: Introduction*, Springer, New York, 1997.

Dantzig, George B., *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

Derman, Cyrus, *Finite State Markov Decision Processes*, Academic Press, New York, NY, 1970.

Drake, A, *Observation of a Markov Process Through a Noisy Channel*, Sc.D. Thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1962.

Eagle, James N., "The Optimal Search for a Moving Target When the Search Path is Constrained," *Operations Research,* Vol. 32, No. 5 (1984), pp. 1107-1115.

Eagle, James N., and Lynn C. Thomas, "Criteria and Approximate Methods for Path-Constrained Moving-Target Search Problems," *Naval Research Logistics,* Vol. 42 (1995), pp. 27-38.

Eckles, James E., "Optimum Maintenance with Perfect Information," *Operations Research*, Vol. 16 (1968), pp. 1058-1067.

Ehrenfeld, S., "On a Sequential Markovian Decision Procedure with Incomplete Information," *Computers and Operations Research*, Vol. (1976), pp. 39-48.

Evans, Dennis K., "Bomb Damage Assessment and Sortie Requirements," *Military Operations Research,* Vol. 2, No. 1 (1996), pp. 31-36.

Fisher, Marshall L., "An Applications-Oriented Guide to Lagrangian Relaxation," *Interfaces,* Vol. 15, No. 2 (1985), pp. 10-21.

Garey, Michael R., and David S. Johnson, *Computers and Intractability; A Guide to the Theory of NP Completeness*, W. H. Freeman and Company, New York, NY, 1979.

Geoffrion, Arthur M., and R. F. Powers, "Twenty Years of Strategic Distribution System Design: An Evolutionary Perspective," Interfaces, Vol. 25, No. 5 (1995), pp. 105-127.

Gilmore, P. C., and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem," *Operations Research*, Vol. 9 (1961), pp. 849-859.

Gilmore, P. C., and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem — Part II," *Operations Research*, Vol. 11 (1963), pp. 863-888.

Greenburg, Harvey J., "Post-Solution Analysis in LP from an Interior Solution," presentation, Sixth INFORMS Computer Science Technical Section Conference, Monterey, CA, February 1998.

Grundhauser, Larry, Susan Mashiko, Hugh Hortsman, and Rick Anderson, "The Future of BDA," in *Concepts in Airpower for the Campaign Planner*, Air Command and Staff College, Maxwell AFB, Al, 1993, pp. 85-106.

Hauskrecht, Milos, "Incremental Methods for Computing Bounds in Partially Observable Markov Decision Processes," in *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, RI, 1997, pp. 734-739.

Hauskrecht, Milos, *Planning and Control in Stochastic Domains with Imperfect Information*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1998.

Higle, Julia, "Modeling in Stochastic Programming Using a Case Study Approach," presentation, VII International Conference on Stochastic Programming, University of British Columbia, Vancouver, CA, August 1998.

Hillier, Frederick S., and Gerald J. Lieberman, *Introduction to Operations Research*, Holden-Day, Oakland, CA, 1986.

Holder, Allen G., "Analyzing the Central Path and Analytic Center Solution with Respect to Data Pertubations," presentation, Sixth INFORMS Computer Science Technical Section Conference, Monterey, CA, February 1998.

Howard, Ronald E., *Dynamic Programming and Markov Processes,* MIT Press, Cambridge, Massachusetts, 1960.

IBM Corporation, *Optimization Subroutine Library Guide and Reference*, Release 2, Document No. SC23-0519-03, Kingston, New York, 1992.

Iglehart, Donald L., "Optimality of (s,S) Policies in the Infinite Horizon Dynamic Inventory Problem," *Management Science*, Vol. 9 (1963), pp. 259-267.

ILOG, Inc., *Using the CPLEX Callable Library*, Incline Village, NV, 1997.

Jonsbraten, Tore W., *Optimal Selection and Sequencing of Oil Wells under Reservoir Uncertainty*, technical report, Department of Business Adminstration, Stavanger College, Norway, 1997.

Jonsbraten, Tore W., Roger J-B Wets, and David L. Woodruff, *A Class of Stochastic Programs with Decision Dependent Random Elements*, technical report, University of California Davis, 1997.

Kall, Peter, and Stein W. Wallace, *Stochastic Programming*, John Wiley and Sons, New York, NY, 1994.

Karmarkar, Narendra K., "A New Polynomial-Time Algorithm for Linear Programming," *Combinatorics*, Vol. 4 (1984), pp. 373-395.

Lane, Daniel E., "A Partially Observable Model of Decision Making by Fishermen," *Operations Research*, Vol. 37, No. 2 (1989), pp. 240-254.

Larsen, Richard J., and Morris L. Marx, *An Introduction to Mathematical Statistics and its Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

Lewis, Richard B. H., "JFACC Problems Associated with Battlefield Preparation in DESERT STORM," *Airpower Journal,* Vol. 8, No. 1 (1994), pp. 4-21.

Littman, Michael L., *The Witness Algorithm for Solving Partially Observable Markov Decision Processes*, Technical Report CS-94-40, Brown University, Providence, RI, 1994.

Lovejoy, William S., "A Survey of Algorithmic Methods for Partially Observed Markov Decision Processes," *Annals of Operations Research*, Vol. 28, No. 1 (1991), pp. 47-65.

Lovejoy, William S., "Computing Feasible Bounds for Partially Observed Markov Decision Processes," *Operations Research,* Vol. 39, No. 1 (1991), pp. 162-175.

Lovejoy, William S., "On the Convexity of Policy Regions in Partially Observed Systems," *Operations Research*, Vol. 35, No. 4 (1987), pp. 619-621.

MacQueen, J., "A Test for Suboptimal Actions in Markovian Decision Problems," *Operations Research*, Vol. 15, No. 3 (1967), pp. 559-561.

Manor, G., and Moshe Kress, *Optimality of the Greedy Shooting Strategy in the Presence of Incomplete Damage Information*, Working Paper, CEMA — Center for Military Analysis, Ministry of Defense/Armament Development Authority, Israel, 1996.

Marshall, Kneale T., and Robert M. Oliver, *Decision Making and Forecasting*, McGraw-Hill, New York, 1995.

Mattheiss, T. H., "An Algorithm for Determining Irrelevant Constraints and all Verticies in Systems of Linear Inequalities," *Operations Research*, Vol. 21 (1973), pp. 247-260.

Meuleau, Nicolas, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaebling, Thomas Dean, and Craig Boutilier, "Solving Very Large Weakly Coupled Markov Decision Processes," in *Proceedings of AAAI-98*, 1998, pp. 165-172.

Microsoft Corporation, *Visual Basic Programmer's Guide,* Document No. DD93011-1296, 1997.

Might, Robert, "Decision Support for Aircraft and Munitions Procurement," *Interfaces*, Vol. 17, No. 5 (1987), pp. 55-63.

Monahan, George E., "A Survey of Partially Observable Markov Decision Processes," *Management Science*, Vol. 28, No. 1 (1982), pp. 1-16.

Monahan, George E., *On Optimal Stopping in a Partially Observable Markov Chain with Costly Information,* Ph.D. dissertation, Northwestern University, 1977.

Monahan, George, E., "Optimal Stopping in a Partially Observable Markov Chain with Costly Information," *Operations Research,* Vol. 28, No. 6 (1980), pp. 1319-1334.

Murkejee, Sraban and Kiran Seth, "A Corrected and Improved Computational Scheme for Finite Horizon Partially Observable Markov Decision Processes, *INFOR,* Vol. 29, No. 3 (1991), pp. 206-212.

Papadimitriou, Christos H., and John N. Tsitsiklis, "The Complexity of Markov Decision Processes," *Mathematics of Operations Research*, Vol. 12 (1987), pp. 441-450.

Parker, R. Gary, and Ronald L. Rardin, *Discrete Optimization,* Academic Press, San Diego, 1988.

Puterman, Martin L., *Markov Decision Processes — Discrete Stochastic Dynamic Programming*, John Wiley and Sons, New York, NY, 1994.

Raffensburger, John R., personal communication, April 1998.

Reed, C. Christopher, *Air Attacks vs. Fixed Defended Ground Targets: Combat Models with Imperfect, Non-Instantaneous BDA,* annotated briefing, The Aerospace Corporation, Los Angeles, 1996.

Rice, Roy E., *Mathematical Formulation of the Sensor-Platform Allocation Model (SPAM),* Working Paper, Teledyne-Brown Engineering, Huntsville, AL, 1997.

Ross, Sheldon M., "Quality Control Under Markovian Deterioration," *Management Science,* Vol. 17, No. 9 (1971), pp 587-596.

Ross, Sheldon M., *Introduction to Probability Models*, Academic Press, Boston, Mass., 1993.

Scott, William B., "Computer/IW Efforts Could Shortchange Aircraft Programs," *Aviation Week and Space Technology*, January 19, 1998, p. 59.

Sondik, Edward J., *The Optimal Control of Partially Observable Markov Processes*, Ph.D. dissertation, Stanford University, Palo Alto, CA, 1971.

Striebel, Charlotte T., "Sufficient Statistics in the Optimal Control of Stochastic Systems," *Journal of Mathematical Analysis and Applications*, Vol. 12 (1965), pp. 576-592.

Wallace, Stein W., "Introduction and Overview of Stochastic Programming," presentation, VII International Conference on Stochastic Programming, University of British Columbia, Vancouver, CA, August 1998.

White, Chelsea C. III, "Optimal Inspection and Repair of a Production Process Subject to Deterioration," *Journal of the Operational Research Society*, Vol. 29 (1978), pp. 235-243.

White, Chelsea C. III, and William T. Scherer, "Finite-Memory Suboptimal Design for Partially Observed Markov Decision Processes," *Operations Research,* Vol. 42, No. 3 (1989), pp. 439-455.

White, D. J., "A Survey of Applications of Markov Decision Processes," *Journal of the Operational Research Society,* Vol. 44, No. 11 (1993), pp. 1073-1096.

White, D. J., "Further Real Applications of Markov Decision Processes," *Interfaces,* Vol. 18, No. 5 (1988), pp. 55-61.

White, D. J., "Real Applications of Markov Decision Processes," *Interfaces,* Vol. 15, No. 6 (1985), pp. 73-78.

White, Franklin E., "Forward," in *Multisensor Data Fusion* by Waltz, Edward, and James Llinas, Artech House, Boston, 1990, pp. xi-xiii.

Willstatter, Kurt, and James Barnes, "Intelligence, Surveillance, and Reconnaissance Investment Study," presentation, 66th Military Operations Research Society Symposium, Monterey, CA, June 1998.

Yost, Kirk A., "Consolidating the USAF's Conventional Munitions Models," *Military Operations Research,* Vol. 2, No. 4 (1996), pp. 53-72.

INITIAL DISTRIBUTION LIST

No. of Copies

1. Defense Technical Information Center 2
   8725 John. J. Klingman Rd., Ste 0944
   Ft. Belvor, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Dr. Peter Purdue Code 08 1
   Naval Postgraduate School
   Monterey, CA 93943-5002

4. Dr. Alan R. Washburn Code OR/Ws 10
   Naval Postgraduate School
   Monterey, CA 93943-5002

5. Dr. Gerald G. Brown Code OR/Bw 1
   Naval Postgraduate School
   Monterey, CA 93943-5002

6. Dr. Robert F. Dell Code OR/De 1
   Naval Postgraduate School
   Monterey, CA 93943-5002

7. Dr. Guillermo Owen Code MA/Ow 1
   Naval Postgraduate School
   Monterey, CA 93943-5002

8. Dr. Craig Rasmussen Code MA/Ra 1
   Naval Postgraduate School
   Monterey, CA 93943-5002

9. Dr. R. Kevin Wood Code OR/Wd 1
   Naval Postgraduate School
   Monterey, CA 93943-5002

10. Dr. Richard E. Rosenthal Code OR/Rl 1
    Naval Postgraduate School
    Monterey, CA 93943-5002

11. Dr. David P. Morton 1
Department of Mechanical Engineering
Engineering Teaching Center
The University of Texas at Austin
Austin, TX 78712

12. Dr. Anthony R. Cassandra 1
Microelectronics and Computer Technology Corporation
3500 West Balcones Center Dr.
Austin, TX 78759-5398

13. Dr. Leslie Pack Kaebling 1
Computer Science Department, Box 1910
Brown University
Providence, RI 02912-1210

14. Dr. Nevin L. Zhang 1
Computer Science Department
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

15. AFIT/CIGG Bldg 125 1
2950 P Street
Wright-Patterson AFB, OH 45433-7765

16. AFOSR/NM 1
Attn: Dr. Neal Glassman
110 Duncan Ave, Ste. 100
Bolling AFB, Washington DC 20332-0001

17. AFIT/ENS Bldg 640 1
Attn: Major Ray Hill
2950 P Street
Wright-Patterson AFB, OH 45433-7765

18. Air University Library (AU/LD) 1
600 Chennault Circle
Maxwell AFB, AL 36112-5001

19. AFSAA/SAN 1
1570 Air Force Pentagon
Washington, DC 20330-1570

20. LtCol Steven F. Baker 1
Department of Management
2354 Fairchild Drive, Ste. 6H94
USAF Academy, CO 80840-5701

21. LtCol Kirk A. Yost 10
    Forces Division
    Force Structure and Analysis Directorate
    The Joint Staff/J-8
    8000 Joint Staff Pentagon
    Washington, DC 20318-8000

22. Col Dan McCorry 5
    DARPA
    3701 N. Fairfax Drive
    Arlington, VA 22203-1714

23. Dr. Roy E. Rice 1
    Teledyne Brown Engineering
    Cummings Research Park
    P.O. Box 070007
    Huntsville, AL 35807-7007

24. Dr. Steven M. Robinson 1
    386 Mechanical Engineering Building
    1513 University Avenue
    Madison, WI 53706-1572

25. Michael Daugherty 1
    J53 Weapons Effects
    Joint Warfare Analysis Center
    18385 Frontage Road
    Dahlgren, VA 22448-5500

26. Dr. Bob Sheldon 1
    System Simulation Solutions, Inc.
    1700 Diagonal Road, Suite 310
    Alexandria, VA 22314

27. Dr. Shelby Brumelle 1
    Faculty of Commerce
    University of British Columbia
    Vancouver, BC V6T 1Y8 Canada

28. Dr. David Castenon 1
    Department of Electrical and Computer Engineering
    8 St. Mary's Street
    Boston, MA 02215

29. OAS/XR 1
    3550 Aberdeen Avenue SE
    Kirtland AFB, NM 87117-6008